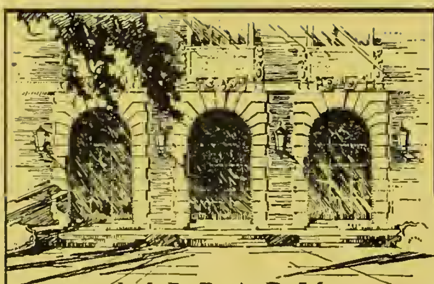

Report No. 329

RANDOM NUMBER GENERATION ON THE I.B.M. 360

by

Beth Carson Richardson

April, 1969

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

RANDOM NUMBER
GENERATION ON THE I.B.M. 360

BY

BETH CARSON RICHARDSON

B.S., University of Illinois, 1966

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois, 1969

Urbana, Illinois

ERRATA

RANDOM NUMBER GENERATION ON THE I.B.M. 360
(Report No. 329 - April, 1969)

by

Beth Carson Richardson

Page	Line	THE LINE NOW READS	THE LINE SHOULD READ
7	19	[Now if	Now if
	21	$(d,m) = g \quad 1$	$(d,m) = g \neq 1$ and $dx \equiv dy (\mathrm{mod}\ m)$ then $x \equiv y (\mathrm{mod}\ m/g)$.
8	1	$x_1 \equiv ax_o (\mathrm{mod}\ m) =$	$x_1 \equiv ax_o (\mathrm{mod}\ m)$
10	17	$= 3^{2^s} + 2^s \ 8k [$	$= 3^{2^s} + 2^s \cdot 8k [$
14	17	$i = 1, \ldots k$	$i = 1, \ldots, k$
15	1	$s_1, \ldots, s_k$ in	$s_1, \ldots, s_{k^m}$ in
	11	$x_2, \ldots, x_{m+1})$	$(x_2, \ldots, x_{m+1})$
17	14	$X^2{}_j = 8/N \sum_{i=0}^{7}$	$X^2{}_j = (8/N) \sum_{i=0}^{7}$
	26	$X^2{}_2 = 100/N \sum_{i,j=1}^{10}$	$X^2{}_2 = (100/N) \sum_{i,j=1}^{10}$
	26	$X^2{}_1 = 10/N \sum_{i,j=1}^{10}$	$X^2{}_1 = (10/N) \sum_{i,j=1}^{10}$

Page	Line	THE LINE NOW READS	THE LINE SHOULD READ
18	2	$X^2_2 - X$	$X^2_2 - X^2_1$
	22	$\theta_1^{a_1} \ldots \theta_k^{a_k}$	$\theta_1^{a_1} \ldots \theta_k^{a_k}$ where $\sum_{i=1}^{k} a_i = n$
	24	$= 8 \frac{5!}{5!(0!\cdots 0!)} (1/8)^5 [(1/8)^0 \cdots (1/8)^0]$	$= 8\left[\frac{5!}{5!(0!\cdots 0!)} (1/8)^5 [(1/8)^0 \cdots (1/8)^0]\right]$
22	9	$x_{i+k-1} < 1/2$	$x_{i+k-1} \leq 1/2$
24	3	$\left[P(x_{i+1} > a) \quad P(x_{i+n} > a)\right]$	$\left[P(x_{i+1} > a) \cdots P(x_{i+n} > a)\right]$
26	3	and, most	and most
26	4	important, it	important it
27	2	pseudorandom	pseudo random
37	TEST 7	.32 .96 .93 .61 .20 .51 .99 .99 .99 .99 .99 .99 .99 .99	.32 .96 .93 .61 .20 .51 .99 .99 .99 .99 .99 .99 .99 .99 ϵ ϵ
43	5	$(0,2^{22})$.	$(0,2^{24})$.
47	12	(N=RN=NTUPLE=ISOC=CHISQ=PROB)	(N,RN,NTUPLE,ISØC,CHISQ,PRØB)

Page	Line	THE LINE NOW READS	THE LINE SHOULD READ
47	18	ISOC - If ISOC = 1 then the N random numbers are to be divided into N/NTUPLE consecutive, non-overlapping, tuples of numbers.	ISØC - If ISØC = 1 then the N random numbers are to be divided into N-NTUPLE+1 successive tuples of numbers. If ISØC = 2 then the N random numbers are to be divided into N/NTUPLE consecutive, non-overlapping, tuples of numbers.
53	16	NX > N.	NX $\geqslant$ N.
61	3	$X_1, \ldots X_N$	$X_1, \ldots, X_N$
	18	$1, \ldots m$	$1, \ldots, m$
	22	$1, \ldots m$	$1, \ldots, m$
64	4	$x_1, \ldots x_N$	$x_1, \ldots, x_N$
	20	$1, \ldots m$	$1, \ldots, m$
69	7	$\surd 2X^2$	$\sqrt{2X^2}$

ACKNOWLEDGMENT

The author is very grateful to Professor L. D. Fosdick for his encouragement and guidance during the preparation of this thesis. The computational work described herein was performed on the I.B.M. 360 50-75 System at the University of Illinois. This research was supported by TR US AEC Contract 1469.

TABLE OF CONTENTS

HISTORY OF RANDOM NUMBER GENERATORS

Random numbers have been traditionally used in simulation studies but have also proven useful for solving problems in numerical analysis. The Monte Carlo method is the name given to any such calculations involving random numbers. Though the term "Monte Carlo" was apparently first used by Von Neumann[1] in 1946, the idea of using random numbers in sampling experiments can be traced back to Student[2] in 1908. Then, and for some time later, random numbers were generated by haphazard techniques. The first random numbers were obtained by drawing cards from a well-shuffled deck, throwing coins, or rolling dice. In an effort to reduce the human bias which was evident in such methods, Tippett collected numbers from census reports and published a table of 40,000 digits[3] in 1925. Kendall and Babington-Smith tested Tippett's numbers and found them inadequate when the same set of numbers was used over and over again in a sampling experiment. In order to supplement Tippett's tables they published a new series of 100,000 random numbers[4] produced by reading digits from a spinning disk when it was illuminated by a flash lamp. Then in the late 1940's computers began to be used to solve Monte Carlo problems and so a need arose for a larger supply of random numbers of high quality. The Rand Corporation filled this need with a table of a million digits[5] generated by monitoring a random frequency pulse source. This table was intended for use by punched card machines. However, computers developed in sophistication and it became impractical for a high-speed electronic device to read random numbers from storage. It was more desirable to have the computer

manufacture its own numbers as needed by a simple arithmetic process. Of course, numbers produced in such a deterministic fashion are not "truly" random and so arithmetically generated numbers that manage to pass statistical tests for randomness are called "pseudo" random.

Arithmetic methods are based on a recurrence relation $x_{i+1} = f(x_i,\ldots,x_{i-n})$, $i-n = 0$, $n = 0,1,\ldots$ in which each new number is generated from the previous ones in such a way that the numbers appear to be drawn at random from the finite population of all the numbers the computer can produce. Because of the nature of this recurrence relation, whenever a set of n+1 numbers is generated that was previously generated, the numbers produced from then on will be identical with the numbers produced after the previous occurrence of the set of n+1 numbers. Thus there is a sequence in which the numbers are formed that will repeat continuously and the length of this sequence is called the period. Starting values $x_0,\ldots,x_n$ are needed to initialize the recurrence relation and herein lies one of the advantages of arithmetic generators. When an arithmetic generator is used, completely new sets of numbers can be produced by changing the starting values, but when a table of random numbers is used the same digits must be read again and again for each new problem. Arithmetic generators are superior to previously-mentioned methods in other ways too. The arithmetic methods are faster, the generated numbers are reproducible so that calculations can be identically repeated, and the numbers can be analyzed for their theoretical properties like length of period. It is no wonder then that numerous arithmetic generators have been proposed and used since Von Neumann first introduced the mid-square method in 1946. In order to select the most appropriate one of these methods three properties of the generated numbers should be considered:

their randomness, the length of the period, and the generation time needed.

In the mid-square method[6] a random number is produced by taking the middle n digits of the square of the previous n digit number. This new random number is then squared and the process continues. One drawback of this method is that a number may occur that will be endlessly repeated. For example, suppose $x_n = 3500$. Then

$$x_n^2 = 12{,}250{,}000 \quad ; \quad x_{n+1} = 2500$$

$$x_{n+1}^2 = 6{,}250{,}000 \; ; \quad x_{n+2} = 2500$$

Other drawbacks are that the method is slow and the period is short.

For these reasons, the mid-square method was superseded by the congruential method. The congruential method is based on the relation $x_{i+1} \equiv ax_i+c \pmod{m}$ which means that ax_i+c is to be divided by m and x_{i+1} set equal to the remainder. Lehmer[7] first suggested this method in 1949 with c=0 in which case it is called the multiplicative congruential method. The mixed congruential method with $c \neq 0$ was later introduced by Rotenberg.[8] The modulus m, the starting value x_0, the multiplier a and the constant c are chosen so as to provide a long period and good statistical behavior. The problem of choosing constants is discussed in section two. Since x_{i+1}, $i=0,1,\ldots$ is equal to the remainder of ax_i+c upon division by the modulus m, each generated number lies somewhere in the range $(0,1,\ldots,m-1)$ so the period of a generator is less than or equal to m. In this regard the mixed method has an apparent advantage over the multiplicative method because for a suitable choice of constants the full period m can be achieved whereas the full period can never be achieved for the multiplicative

method. Unfortunately, extensive testing[9,10] has shown that the multiplicative method generally behaves better statistically than the mixed method. The poor statistical behavior of the mixed method led Hull and Dobell[11] to conclude that multiplicative methods were to be preferred.

In an effort to speed up congruential methods even more, the multiplication operation was replaced by an addition so that the recurrence relation became $x_{i+1} \equiv x_i + x_{i-n} (\text{mod } m)$. This relation is known as the additive method and when $n = 1$, $x_0 = 0$ and $x_1 = 1$ one has the Fibonacci generator. Although the Fibonacci generator is faster and has a longer period than the multiplicative method, testing has shown that it gives poorer statistical results. When n was increased the generator produced numbers with better statistical properties but the program required indexing so the speed advantage of the additive method was lost.[12]

MacLaren and Marsaglia[13] found mixed congruential generators to have poor statistical qualities when used in Monte Carlo calculations and so proposed two improved congruential generators. One of the generators uses a stored table of random numbers. When a number is used it is replaced by a scrambled version of itself provided by the relation $x_i \equiv ax_i + c (\text{mod } m)$. This method eliminates the main problem of stored tables of numbers, that of exhausting the list, but requires some inconvenient tape-handling. The second generator is designed to improve the distribution of n-tuples of random numbers $x_1, \ldots, x_n$. A list of 128 numbers $u_1, \ldots, u_{128}$ is produced by the relation $u_{i+1} \equiv a_1 u_1 + c_1 (\text{mod } m)$. A second set of numbers $v_1, \ldots, v_n$ where $v_{i+1} \equiv a_2 v_i + c_2 (\text{mod } m)$ is also needed. The first seven bits of v_k are used to select the k'th random

number, x_k, from the list of the u's. This position is then refilled with v_{k+1}. Although the numbers produced by this generator were shown to have better statistical properties than numbers produced by the mixed or multiplicative methods, this generator has other problems. Its theoretical properties are unknown and the generation time is about twice that of the mixed or multiplicative methods because two numbers must be produced to generate every one random number.

In conclusion, it appears that there is no generator that provides random numbers with the fastest generation time, the longest period, and the best statistical properties. The generator which appears best to combine these three properties, though, is the multiplicative congruential generator.

FOOTNOTES

1. R. D. Richtmyer, "Monte Carlo Methods," Nuclear Reactor Theory Am. Math. Soc., Proc. Symposium Appl. Math. 11 (1961), pp. 190-205.

2. Student, "The Probable Error of a Mean," Biometrika 6 (1908), pp. 1-25.

3. L. H. C. Tippett, Random Sampling Numbers, Tracts for Computers, no. 15 (Cambridge, 1927).

4. M. G. Kendall and B. Babington-Smith, Tables of Random Sampling Numbers, Tracts for Computers, no. 24 (Cambridge, 1939).

5. Rand Corporation, A Million Random Digits with 100,000 Normal Deviates, (Free Press, Glencoe, 1955).

6. B. Jannson, Random Number Generators, (Victor Pettersons Bokindustri Aktiebolag, Stockholm, 1966), p. 31.

7. D. H. Lehmer, "Mathematical Models in Large-Scale Computing Units," Annals Comp. Laboratory, Harvard Univ. 26 (1951), pp. 141-146.

8. A. Rotenberg, "A New Pseudo-Random Number Generator," J. Assoc. Comp. Mach. 7 (1960), pp. 75-77.

9. T. E. Hull and A. R. Dobell, "Mixed Congruential Random Number Generators for Binary Machines," J. Assoc. Comp. Mach. 11 (1964), pp. 31-40.

10. M. D. MacLaren and G. Marsaglia, "Uniform Random Number Generators," J. Assoc. Comp. Mach. 12 (1965), pp. 83-88.

11. Hull and Dobell, J. Assoc. Comp. Mach. 11.

12. B. F. Green, Jr., J. E. Smith, and L. Klem, "Empirical Tests of an Additive Random Number Generator," J. Assoc. Comp. Mach. 6 (1959) pp. 527-536.

13. MacLaren and Marsaglia, J. Assoc. Comp. Mach. 12.

THEORETICAL PROPERTIES OF THE MULTIPLICATIVE CONGRUENTIAL METHOD

The congruence relation for the multiplicative method, $x_{i+1} \equiv ax_i (\mathrm{mod}\ m)$, requires the programmer to define three parameters: the modulus m, the starting value x_0, and the multiplier a. If chosen wisely, these parameters will provide speed, a long period and good statistical properties. The following paragraphs describe some rules by which the parameters should be chosen if these goals are to be attained.

The modulus m is usually chosen to be 2^r where r is a positive integer less than or equal to the number of bits in a computer word.[1] This choice provides a speed advantage for suppose that

$$ax_i = b_{r+n}2^{r+n}+\ldots+b_r 2^r+b_{r-1}2^{r-1}+\ldots+b_1 2^1+b_0 \text{ , } b = 0 \text{ or } 1.$$

Now $x_{i+1} \equiv ax_i (\mathrm{mod}\ m)$ means that ax_i is to be divided by m and x_{i+1} set equal to the remainder, but if $m = 2^r$ then the division is eliminated because reduction modulo m can be achieved by merely setting to zero all but the low order r bits of ax_i.

The starting value x_0 should be chosen relatively prime to the modulus m or the period may be reduced.[2] The proof of this statement requires the following fact from number theory. Let $(d,m) = g$ represent the fact that the greatest common divisor of d and m is g. [Now if $(d,m) = 1$, that is d and m are relatively prime, and $dx \equiv dy (\mathrm{mod}\ m)$ then $x \equiv y (\mathrm{mod}\ m)$ if $(d,m) = g \quad 1$ and $dx \equiv dy (\mathrm{mod}\ m)$ then $x \equiv y (\mathrm{mod}\ m/g)$]. Suppose $(x_0,m) = 1$ and the period is n.

$x_1 \equiv ax_0 (\text{mod } m) =$

$x_2 \equiv ax_1 (\text{mod } m) = a^2 x_0 (\text{mod } m)$

$\vdots$

$x_n \equiv a^n x_0 (\text{mod } m)$

Since x_n is the n+1'st number in the sequence it is equal to x_0. So $a^n x_0 \equiv x_0 (\text{mod } m)$ or $a^n \equiv 1 (\text{mod } m)$. Now suppose $(x_0, m) = g \neq 1$ and the period is n

$x_n \equiv a^n x_0 (\text{mod } m)$

$a^n x_0 \equiv x_0 (\text{mod } m)$

$a^n \equiv 1 (\text{mod } m/g)$

The period may be shortened because the modulus has been reduced by the factor g. The proof of this statement hinges on this fact: If n is the period modulo m then it is the period modulo m/g, but the reverse is not always true. Suppose $a^n \equiv 1 (\text{mod } m)$, then $a^n = \lambda \cdot m+1$ where λ is an integer or $a^n = \lambda \cdot g \cdot m/g+1$ so $a^n \equiv 1 (\text{mod } m/g)$. Now suppose $a^n \equiv 1 (\text{mod } m/g)$, then $a^n = \lambda \cdot m/g+1$ and $a^n \equiv 1 (\text{mod } m)$ if and only if λ/g is an integer. This fact proves that the period when $(x_0, m) \neq 1$ will be the same or less than the period when $(x_0, m) = 1$. If the modulus m is chosen to be 2^r, the requirement on x_0 reduces to choosing x_0 such that it is any odd positive integer.

The multiplier a should be chosen so that it provides random numbers with the longest period and the best statistical properties. Greenberger[3] showed that the longest period is obtained when $a = 8k \pm 3$ where k is any positive integer. The following results from number theory[4] are needed to explain this rule.

(1) The Euler Phi function $\varphi(m)$ is defined as the number of positive integers less than m and relatively prime to m.

(2) Euler's theorem states that if $(a,m) = 1$ then $\varphi(m)$ is the largest integer such that $a^{\varphi(m)} \equiv 1(\text{mod } m)$.

(3) If the least positive integer n such that $a^n \equiv 1(\text{mod } m)$ is such that $n = \varphi(m)$ then a is called a primitive root of m.

(4) If $n < \varphi(m)$ then n is a divisor of $\varphi(m)$.

As was shown in the preceding paragraph the period is the smallest positive integer n such that $a^n \equiv 1(\text{mod } m)$ where we will assume that the modulus m is 2^r. We would like the smallest n to be as large as possible. Now if a is a primitive root of 2^r then $n = \varphi(2^r)$ by (3) and if in addition a is odd then this n will be maximum by (2). However, the only moduli which have primitive roots are 4, $2\cdot p^\alpha$, and p^α for p and odd prime so since 2^r cannot have a primitive root $n < \varphi(2^r)$. Now $\varphi(2^r) = 2^{r-1}$ by (1) and since a is not a primitive root n must be of the form 2^s for $s \leq r-2$ by (4). The theory requires that a be an odd positive integer and all odd positive integers can be written in the form $8k\pm1$ or $8k\pm3$ for k a non-negative integer. We would like to know which form of a will give the maximum period.

Suppose that a is $8k\pm1$. The period is 2^s for $s \leq r-2$ and to determine the period we want to find the minimum s satisfying

$$(8k\pm1)^{2^s} \equiv 1(\text{mod } 2^r)$$

$$\text{or } (8k\pm1)^{2^s} - 1 \equiv 0(\text{mod } 2^r)$$

This means that $(8k\pm1)^{2^s} - 1$ is evenly divisible by 2^r.

Expanding by the binomial theorem,

$$(\pm1+8k)^{2^s} - 1 = \pm2^s\cdot 8k + \frac{2^s(2^s-1)(8k)^2}{2!} \pm \cdots$$

$$= (2^s\cdot 8k)\left[\pm1 + \frac{(2^s-1)\,8k}{2} \pm \cdots\right]$$

All the terms in square brackets that include an 8k will be even as long as one of the denominators does not cancel out a numerator. The fact that this can never happen is seen by setting $n = 2^s$ and rewriting the j+1st term. We have

$$\left\{ \frac{(n-1)(n-2)\cdots(n-j+1)}{j!} \right\} \cdot (8k)^j$$

The part in curly brackets is the binomial coefficient $\binom{n-1}{j}$ and every binomial coefficient is an integer. Therefore every term in square brackets except the first term will be even so the contents of the square brackets is odd. Since 2^r cannot evenly divide an odd number it must be true that $2^s \cdot 8k \equiv 0 (\mathrm{mod}\ 2^r)$ or $2^{s+3} \cdot k \equiv 0 (\mathrm{mod}\ 2^r)$. The smallest s satisfying this relation depends on the choice of k and is such that $s \leqslant r-3$. So the maximum period for a multiplier of the form $8k \pm 1$ is 2^{r-3}.

Now suppose instead that a is $8k \pm 3$. We want to find the smallest s such that $(8k \pm 3)^{2^s} \equiv 1 (\mathrm{mod}\ 2^r)$. Again expand by the binomial theorem

$$(\pm 3+8k)^{2^s} = 3^{2^s} \pm 2^s \cdot 8k \cdot 3^{2^s -1} + \frac{2^s(2^s-1)(8k)^2 \cdot 3^{2^s -2}}{2!} \pm \cdots$$

$$= 3^{2^s} + 2^s\ 8k \left[\pm 3^{2^s -1} + \frac{(2^s-1)(8k)\ 3^{2^s -2}}{2} \pm \cdots \right]$$

Using the same reasoning as for the $8k \pm 1$ case, all terms in square brackets except the first will be even so the contents of the square brackets is odd.

Then $(\pm 3+8k)^{2^s} = 2^{s+3} \cdot k \cdot [\mathrm{odd}] + 3^{2^s}$

But $3^{2^s} = (-1+2^2)^{2^s}$ and expanding this by the binomial theorem

$$3^{2^s} = 1-2^s \cdot 2^2 + \frac{2^s(2^s-1)(2^2)^2}{2!} - \ldots$$

$$= 1-2^{s+2}\left[1-\frac{(2^s-1)(2^2)}{2} + \ldots\right]$$

The contents of the square brackets is again odd.

$$= 1-2^{s+2}[\text{odd}]$$

Now $(8k\pm3)^{2^s} - 1 \equiv 0(\text{mod } 2^r)$

But $(8k\pm3)^{2^s} - 1=-2^{s+2}[\text{odd}] + 2^{s+3}\cdot k\cdot[\text{odd}]$

$$=2^{s+2}(-[\text{odd}] + 2k\cdot[\text{odd}])$$

$$=2^{s+2}[\text{odd}]$$

Since 2^r cannot evenly divide an odd term, it must divide the 2^{s+2} term. The smallest s satisfying this is s = r-2. Therefore the period for a multiplier of the form $8k\pm3$ is 2^{r-2}. This proves that the longest period for the multiplicative congruential method is obtained when the multiplier a is of the form $8k\pm3$.

Unfortunately, one of the properties of the multiplicative method is that only the most significant bit has the maximum period and the length of period decreases with the significance of the bit. This is explained in the following way. Let $x_{n+1} \equiv ax_n(\text{mod } 2^r)$. If $a = 8k\pm3$ then the period of the x's is of length 2^{r-2}. Suppose

$$ax_n = \ldots b_{r+1}2^{r+1}+b_r2^r+b_{r-1}2^{r-1}+\ldots+b_12^1 + b_02^0$$

$$\text{then } ax_n(\text{mod } 2^r) = b_{r-1}2^{r-1}+\ldots+b_12^1+b_02^0$$

or $x_{n+1} = b_{r-1}2^{r-1} + y_n$ where $y_n \equiv ax_n (\mathrm{mod}\ 2^{r-1})$.

Now the sequence of y's has the period 2^{r-3}, but $y_n = b_{r-2}2^{r-2} + z_n$ where $z_n \equiv ax_n\ (\mathrm{mod}\ 2^{r-2})$ so the sequence of z's has the period 2^{r-4}. Clearly then, b_{r-i} has period 2^{r-i-1}. Finally, since the multiplier and the starting value are odd, each generated number will be odd and b_0 will always be 1.

Although a multiplier of the form $8k\pm3$ will provide random numbers with the longest period, these numbers may not have good statistical properties. Unfortunately, there are no simple rules to ensure the latter and more work needs to be done in this area. From a theoretical standpoint the multiplier should not be close to a simple rational multiple of the modulus m or serial correlation will result.[5] If $a = m/k$ for k an integer then $x_{i+1} \equiv (m/k)x_i (\mathrm{mod}\ m)$ so $x_{i+1} = x_i/k$. Contrary to the recommendation of some authors[6] the multiplier should not be close to the square root of the modulus m either, because this choice results in strong correlation of lag 2. If $a = \sqrt{m}/k$ for k an integer then

$$x_{i+1} \equiv (\sqrt{m}/k)x_i (\mathrm{mod}\ m)$$

$$x_{i+2} \equiv (\sqrt{m}/k)^2\ x_i (\mathrm{mod}\ m) \equiv (m/k^2)x_i (\mathrm{mod}\ m)$$

$$\text{so } x_{i+2} = x_i/k^2$$

Of course no final selection of the multiplier can be approved until a careful check has been made on the generated numbers. For this purpose a series of appropriate statistical tests is discussed in the next section.

FOOTNOTES

1. T. E. Hull and A. R. Dobell, "Random Number Generators," SIAM Review, 4 (1962), pp. 230-254.

2. International Business Machines Corporation, Random Number Generation and Testing, Reference manual C20-8011 (New York, 1959), p. 4.

3. M. Greenberger, "Notes on a New Pseudo-Random Number Generator," J. Assoc. Comp. Mach. 8 (1961), pp. 163-166.

4. O. Ore, Number Theory and Its History, (McGraw-Hill, New York, 1948), pp. 272-302.

5. R. R. Coveyou and R. D. MacPherson, "Fourier Analysis of Uniform Random Number Generators," J. Assoc. Comp. Mach. 14 (1967), pp. 100-119.

6. International Business Machines Corporation, p. 5.

STATISTICAL TESTS

The methods discussed so far are designed to produce uniformly distributed random integers. In addition, a set of uniformly distributed random floating point numbers on the unit interval (0,1) can be produced by interpreting each binary integer as a binary fraction and making the appropriate conversion. Once a supply of uniformly distributed random numbers on the unit interval is available random numbers with other statistical distributions can be easily obtained.[1] The hypothesis to be tested, then, is that the generated numbers are independent and uniformly distributed. For each statistical test the expected results if this hypothesis is true are compared with the actual results by a Chi-Square test.

In order to use the Chi-Square statistic the generated numbers $x_1,\ldots,x_n$ must be separated into distinct classes where the kinds of classes vary with each statistical test. Suppose there are k cells $r_1,\ldots,r_k$ in which the generated numbers can fall and let $x_j \epsilon r_i$ represent the fact that x_j is contained in cell r_i. Let p_i be the expected probability that $x_j \epsilon r_i$ where $j = 1,\ldots,N$; $i = 1,\ldots,k$ and p_i is independent of j. Let f_i be the actual frequency of generated numbers in cell i.

Then $X_1^2 = \sum_{i=1}^{k} (f_i - N\cdot p_i)^2/N\cdot p_i$ has a Chi-Square distribution with k-1 degrees of freedom if the hypothesis of uniform randomness is true.

These results can be extended to more than one dimension. Let $x_j,\ldots,x_{j+m-1}$ be an m sequence of generated numbers. As before, let there be k cells $r_1,\ldots,r_k$ in which an individual number can fall. There are

then k^m cells $s_1,\ldots,s_{k^m}$ in which an m sequence can fall where $(x_j,\ldots,x_{j+m-1}) \epsilon s_i = (r_x, r_y,\ldots,r_z)$ represents the fact that the first member of the sequence fell in cell x, the second member fell in cell y and the m'th member fell in cell z. Let p_i be the expected probability that $x_j,\ldots,x_{j+m-1} \epsilon s_i$, $(j = 1,\ldots,N-m+1;\ i = 1,\ldots,k^m)$ when the hypothesis is true. Let f_i be the actual number of m sequences of generated numbers which are in class i. When the m sequences are taken consecutively, i.e., $(x_1,\ldots,x_m)$, $(x_{m+1},\ldots,x_{2m})$, $(x_{2m+1},\ldots\ )$,... then

$X^2_m = \sum_{i=1}^{k^m} (f_i-[N/m]\cdot p_i)^2/[N/m]\cdot p_i$ has a Chi-Square distribution. However, when the m sequences are taken successively, i.e., $(x_1,\ldots,x_m)$, $(x_2,\ldots, x_{m+1})$, $(x_3,\ldots\ )$,... then the probability that $x_j,\ldots,x_{j+m-1} \epsilon s_i$ is not independent of j so $X^2_m = \sum_{i=1}^{k^m} (f_i-[N-m+1]\cdot p_i)^2/[N-m+1]\cdot p_i$ does not have a Chi-Square distribution. For this case , Good[2] shows that $X^2_m - X^2_{m-1}$ has asymptotically a Chi-Square distribution with $k^m - k^{m-1}$ degrees of freedom. By defining $X^2_0 = 0$, the case for m = 1 holds as before.

Once the Chi-Square statistic X^2 has been calculated the next step is to evaluate $\int_{X^2}^{\infty} f(x)dx$ where

$$f(x) = \frac{x^{n/2-1}e^{-x/2}}{2^{n/2}\Gamma(n/2)}$$ is the Chi-Square probability distribution for n

degrees of freedom. This integral denotes the probability that a Chi-Square random variable will be greater than X^2, or equivalently, denotes the area under the Chi-Square curve to the right of X^2. For degrees of freedom $n \leqslant 30$ the probability can be found in a table of the

Chi-Square distribution. For degrees of freedom $n > 30$ the quantity $\sqrt{2X^2}$ is approximately normally distributed about mean $\sqrt{2n-1}$ with unit variance.[3] A significance level of .05 or .01 is usually selected and if the probability is less than this significance level then the hypothesis of uniform randomness is rejected.

Various statistical procedures that employ such a test have been suggested for testing the randomness of the generated numbers. In 1938 Kendall and Babington-Smith[4] proposed four widely used tests: the frequency test, serial correlation test, poker test, and gap test. Since then a vast number of other tests have also been proposed. Many authors feel that exhaustive testing of a generator is impractical, that it is better to select a generator that satisfies the more straightforward tests like the frequency and serial tests. However, it is the opinion of this author that the greater the number of tests that are passed, the more confidence can be placed in the generated numbers.

One of the most important statistical tests, the frequency test shows how close the generated numbers are to being uniformly distributed. In this test the unit interval is divided into 100 equal cells and the frequency of numbers in each cell f_i, $i = 1,\ldots,100$ is calculated. If the numbers are uniformly distributed then each digit should occur an equal number of times so the probability that a generated number is in any cell is 1/100. For a sample of N numbers the expected frequency of generated numbers in each cell is N/100. The hypothesis of uniform distribution can be tested by comparing the calculated frequencies with the expected frequencies by a Chi-Square test. The statistic $X^2_1 = (100/N) \sum_{i=1}^{100} (f_i - N/100)^2$ has a Chi-Square distribution with

99 degrees of freedom. Note that this procedure tests the uniform distribution of the first two digits of every generated number. It is a much stricter test than the usual procedure of dividing the unit interval into 10 subintervals and thereby only testing the frequency of the first digit of every number.

A frequency test can also be performed on the octal digits of each random integer. Suppose there are m possible octal digit positions in a random integer and let f_{ij} be the frequency of the i'th octal digit in the j'th digit position, $i = 0,\ldots,7$; $j = 1,\ldots,m$. If the integers are uniformly distributed then each octal digit should occur an equal number of times so the expected frequency of each octal digit in each digit position is N/8. A Chi-Square test can be used to compare the expected frequencies with the calculated frequencies. The statistic $X^2_j = (8/N)\sum_{i=0}^{7} (f_{ij}-N/8)^2$ calculated for each of the m octal digit positions $j = 1,\ldots,m$ has a Chi-Square distribution with 7 degrees of freedom.

The serial correlation test determines whether any digit tends to be followed by any other digit. This test is defined by dividing the unit interval into 10 equal subintervals and letting f_{ij} be the frequency of generated numbers in the i'th interval which are followed by a generated number in the j'th interval. Suppose a sample of N generated numbers is tested. If the numbers are truly random then no digit will tend to be followed by any other digit and the frequency of random numbers in each cell will be the same N/100. The expected frequencies can be compared with the calculated frequencies by a Chi-Square test. Let $X^2_2 = (100/N)\sum_{i,j=1}^{10} (f_{ij}-N/100)^2$ and $X^2_1 = (10/N)\sum_{i,j=1}^{10} (f\ -N/10)^2$ where X^2_1

corresponds to a frequency test on the random numbers with 10 divisions of the unit interval. Then $X^2_2 - X^2_1$ has asymptotically a Chi-Square distribution with 90 degrees of freedom. A visual test for serial correlation is derived by noting that the above procedure is equivalent to dividing the unit square into 100 equal cells. If successive pairs of numbers are plotted as points in the unit square then there should be an even scattering of points in all areas of the square. Any preponderance of points in an area represents serial correlation.

The mutual independence of the octal digits in each random integer can be checked by a poker test. To perform this test, the first five octal digits of each integer are treated as a poker hand without regard to suit. Each hand is tallied as either a bust (abcde), one pair (aabcd), two pair (aabbc), three of a kind (aaabc), full house (aaabb), four of a kind (aaaab), or five of a kind (aaaaa). For instance, if the first five octal digits are 62521 this hand is tallied as one pair. The expected frequencies of poker hands assuming uniform distribution and mutual independence can be computed with the use of the multinomial distribution. The multinomial distribution is defined as follows. If there are n independent trials with k outcomes per trial and θ_i, i=1,...,k is the probability of the i'th outcome and a_i, i=1,...,k is the frequency of the i'th outcome in n trials then the joint distribution of $a_1,\ldots,a_k$ is:

$$f(a_1,\ldots,a_k) = \frac{n!}{a_1!\ \ldots\ a_k!}\ \theta_1^{a_1} \ldots \theta_k^{a_k} \quad \text{where } \sum_{i=1}^{k} a_i = n$$

For the poker test n = 5, k = 8, θ_i = 1/8 i=1,...,8

$$p(5 \text{ of a kind}) = 8\left[\frac{5!}{5!(0!\cdots 0!)}\ (1/8)^5\ [(1/8)^0 \cdots (1/8)^0]\right]$$
(aaaaa)

$$p(\text{4 of a kind}) \atop (aaaab) = 8\cdot 7\cdot \left[\frac{5!}{4!1!}(1/8)^4(1/8)^1\right]$$

$$p(\text{Full House}) \atop (aaabb) = 8\cdot 7\cdot \left[\frac{5!}{3!2!}(1/8)^3(1/8)^2\right]$$

$$p(\text{3 of a kind}) \atop (aaabc) = 8\binom{7}{2}\left[\frac{5!}{3!1!1!}(1/8)^3(1/8)^1(1/8)^1\right]$$

$$p(\text{2 pair}) \atop (aabbc) = 8\binom{7}{2}\left[\frac{5!}{2!2!1.}(1/8)^2(1/8)^2(1/8)^1\right]$$

$$p(\text{1 pair}) \atop (aabcd) = 8\binom{7}{3}\left[\frac{5!}{2!1!1!1!}(1/8)^2(1/8)^1(1/8)^1(1/8)^1\right]$$

$$p(\text{bust}) \atop (abcde) = \binom{8}{5}\left[\frac{5!}{1!}(1/8)^5\right]$$

For a sample of random integers of size N the expected frequencies of the 7 poker hands are $N\cdot p_i, i=1,\ldots,7$. The actual frequencies f_i can be compared with the expected frequencies e_i by a Chi-Square test. The statistic $X^2 = \sum_{i=1}^{7} \frac{(f_i - e_i)^2}{e_i}$ has a Chi-Square distribution with 6 degrees of freedom. In the interpretation of this test a positive correlation in the octal digits would result in too many good hands while a negative correlation would result in too many bad hands.

A run test shows how the generated numbers oscillate among themselves. To describe this test suppose we replace the sample of N random numbers $x_1,\ldots,x_N$ by an N-1 sequence $S=s_1,\ldots,s_{N-1}$ where $s_i=0$ if $x_i \leq x_{i+1}$ and $s_i = 1$ if $x_i > x_{i+1}$. Runs can be inside runs or end runs depending on whether they occur inside or at the two ends of the N-sequence. A subsequence of k zeros bounded by ones at each end forms an inside run up of length k. A subsequence of k ones bounded by zeros at each end forms an inside run down of length k. End runs have to be

bounded only on one side. For example, the sequence 001110 begins with a run up of length 2 and is followed by a run down of length 3. The run test involves counting the number of runs of different lengths and comparing them by a Chi-Square test with expected results. In the interpretation of this test a high frequency of long runs indicates non-randomness of the generated numbers. The expected number of runs of length k is:

$$2N\cdot\frac{k^2+3k+1}{(k+3)!} - 2\cdot\frac{k^3+3k^2-k-4}{(k+3)!}$$

This is derived in the following manner.[5] Let r_k be the number of runs of length k.

$$\text{Let } r_{ki} = \begin{cases} 1 & \text{if there is a run of length k starting in the i'th position} \\ & \text{of the S sequence} \\ 0 & \text{otherwise} \end{cases}$$

Then the expected number of runs of length k is:

$E(r_k) = E(\sum_{i=1}^{N} r_{ki})$. Now $E(r_{ki}) = 0$, $i > N-k$ so $E(\sum_{i=1}^{N} r_{ki}) =$

$E(r_{k1}) + \sum_{i=2}^{N-k-1} E(r_{ki}) + E(r_{k,N-k})$. For $2 \leqslant i \leqslant N-k-1$ then

$$E(r_{ki}) = P(10^k1) + P(01^k0) = 2\cdot P(10^k1)$$

$$= 2\cdot P(x_{i-1} > x_i \leqslant x_{i+1} \leqslant \cdots \leqslant x_{i+k-1} \leqslant x_{i+k} > x_{i+k+1})$$

Since $0 < x_j < 1$, then $P(x_j) = \int_0^1 dx_j$

$$P(x_{j-1} \leqslant x_j) = \int_0^{x_j} dx_{j-1}$$

$$P(x_{j-1} > x_j) = \int_{x_j}^{1} dx_{j-1}$$

So $E(r_{ki}) = 2\int_0^1 \int_{x_{i+k+1}}^1 \int_0^{x_{i+k}} \cdots \int_0^{x_{i+1}} \int_{x_i}^1 dx_{i-1} dx_i \cdots dx_{i+k-1} dx_{i+k} dx_{i+k+1}$

$$= 2 \cdot \frac{k^2+3k+1}{(k+3)!}$$

Now $E(r_{k1}) = P(0^k 1) + P(1^k 0) = 2 \cdot P(0^k 1)$

$$= 2 \cdot P(x_1 \leqslant x_2 \leqslant \cdots \leqslant x_k \leqslant x_{k+1} > x_{k+2})$$

$$= 2\int_0^1 \int_{x_{k+2}}^1 \int_0^{x_{k+1}} \int_0^{x_k} \cdots \int_0^{x_2} dx_1 \cdots dx_{k-1} dx_k dx_{k+1} dx_{k+2}$$

$$= 2 \cdot \frac{k+1}{(k+2)!}$$

But since $E(r_{k1}) = E(r_{k,N-k})$ then

$$E(r_k) = E(\sum_{i=1}^{N} r_{ki}) = 2 \cdot E(r_{k1}) + (N-k-2)E(r_{ki})$$

$$= 2N \cdot \frac{k^2+3k+1}{(k+3)!} - 2 \cdot \frac{k^3+3k^2-k-4}{(k+3)!}$$

Another type of run test counts the number of runs above and below the mean. To describe this test suppose we replace the sample of N random numbers $x_1, \ldots, x_N$ by an N sequence $S = s_1, \ldots, s_N$ where $s_i = 0$ if $x_i \leqslant 1/2$ and $s_i = 1$ if $x_i > 1/2$. The number of runs up and down of different lengths are counted as described in the previous paragraphs and compared with expected results by a Chi-Square test. The expected number of runs of length k is $(N-k+3)2^{-k-1}$. This is calculated in the

following manner. Let r_k be the number of runs of length k.

Let $r_{ki} = \begin{cases} 1 \text{ if there is a run of length k starting in the i'th position} \\ \quad \text{of the S-sequence.} \\ 0 \text{ otherwise} \end{cases}$

Then the expected number of runs of length k is:

$E(r_k) = E(\sum_{i=1}^{N} r_{ki})$. Now $E(r_{ki}) = 0,\ i > N-k+1$ so $E(\sum_{i=1}^{N} r_{ki}) =$

$E(r_{k1}) + \sum_{i=2}^{N-k} E(r_{ki}) + E(r_{k,N-k+1})$. For $2 \leqslant i \leqslant N-k$ then

$$E(r_{ki}) = P(10^k1) + P(01^k0) = 2 \cdot P(10^k1)$$

$$= 2 \cdot P(x_{i-1} > 1/2,\ x_i \leqslant 1/2, \ldots, x_{i+k-1} < 1/2,\ x_{i+k} > 1/2)$$

But $P(x_j \leqslant 1/2) = 1/2,\ P(x_j > 1/2) = 1/2$

So $E(r_{ki}) = 2 \cdot P(2^{-k-2}) = 2^{-k-1}$

Now $E(r_{k1}) = P(0^k1) + P(1^k0) = 2 \cdot P(0^k1)$

$$= 2 \cdot P(x_1 \leqslant 1/2, \ldots, x_k \leqslant 1/2,\ x_{k+1} > 1/2) = 2 \cdot (2^{-k-1})$$

$$= 2^{-k}$$

But since $E(r_{k1}) = E(r_{k,N-k+1})$ then

$$E(r_k) = E(\sum_{i=1}^{N} r_{ki}) = 2 \cdot E(r_{k1}) + (N-k-1)E(r_{ki})$$

$$= (N-k+3)2^{-k-1}$$

A test which is similar to the run test is the gap test. If a maximum in a sequence of generated numbers is a number that is surrounded by two smaller numbers and a minimum is a number surrounded by two larger numbers then a gap is defined as the distance between two successive

maxima or minima. In determining the length of a gap the two maxima or minima are included as well as the numbers between them so the smallest gap is of length 3. The gap test is performed by counting the number of gaps of different lengths and comparing them with expected results by a Chi-Square test. For a sample of generated numbers of size N the expected number of gaps of length r is:

$3 \cdot N \cdot \frac{2^{r-1}}{r!} - \frac{r-2}{r+2}$. This formula is derived very clearly by Kermack-McKendrick[6] and it will not be discussed here.

MacLaren and Marsaglia[7] were dissatisfied with the afore-mentioned statistical tests because they thought the tests were irrelevant to actual uses of random numbers. One of the most common ways of using a sequence of random numbers is to sample consecutive n-tuples and so MacLaren and Marsaglia devised frequency tests on the range of n-tuples and on the range of maximums and minimums of n-tuples. If $W_i = \max(x_{i+1}, \ldots, x_{i+n})$ $i = 0, \ldots, N-n$ where the x_i's are independent and uniformly distributed on the unit interval (0,1) then W_i is not uniformly distributed because:

$$\begin{aligned} P(W_i < a) &= P(x_{i+1} < a) \cdots P(x_{i+n} < a) \\ &= \int_0^a dx_{i+1} \cdots \int_0^a dx_{i+n} \\ &= a^n, \end{aligned}$$

whereas uniformity would require $P(W_i < a) = a$. However, $P(W_i < a) = P(W_i^n < a^n) = a^n$ so W_i^n is theoretically uniformly distributed on (0,1). In a sample of N generated numbers there are N/n n-tuples and the observed W_i^n, $i = 1, \ldots, N/n$ can be tested for a uniform distribution by the frequency test as described earlier. In a

similar manner the minimum of n-tuples can be tested. In this case, if $W_i = \min(x_{i+1},\ldots,x_{i+n})$ then

$$P(W_i < a) = 1-P(W_i > a) = 1-\left[P(x_{i+1} > a) \quad P(x_{i+n} > a)\right]$$

$$= 1-\left[\int_a^1 dx_{i+1} \cdots \int_a^1 dx_{i+n}\right]$$

$$= 1-(1-a)^n$$

Now $P(W_i < a) = P\left[1-(1-W_i)^n < 1-(1-a)^n\right] = 1-(1-a)^n$ so $1-(1-W_i)^n$ is theoretically uniformly distributed. A frequency test can again be employed to determine how close the actual results are to being uniformly distributed.

Unfortunately, a random number generator which has passed all of these standard statistical tests may still fail when a particular application is sensitive to a property of the numbers that was not tested. In the end, the most relevant test of a random number generator for a particular application is to use the generator on a similar problem for which the answer is known.

FOOTNOTES

1. B. Jannson, pp. 170-191.

2. I. J. Good, "The Serial Test for Sampling Numbers and Other Tests for Randomness," *Proc. Camb. Phil. Soc.* 49 (1953), pp. 276-284. χ^2

3. M. G. Kendall and A. Stuart, *The Advanced Theory of Statistics*, (Griffin, London, 1958), I, p. 400.

4. M. G. Kendall and B. Babington-Smith, "Randomness and Random Sampling Numbers," *J. Roy. Stat. Soc.* 101 (1938), pp. 147-166. freq. serial poker gap

5. H. Levene and J. Wolfowitz, "The Covariance Matrix of Runs Up and Down," *Annals Math. Stat.* 15 (1944), pp. 58-69. runs

6. W. O. Kermack and A. G. McKendrick, "Some Distributions Associated with a Randomly Arranged Set of Numbers," *Proc. Roy. Soc. Edinburgh* 57 (1937), pp. 332-376. GAP

7. MacLaren and Marsaglia, *J. Assoc. Comp. Mach.* 12. freq. of n-tuples

EXPERIMENTAL RESULTS

The multiplicative method of random number generation has been generally accepted as a good one because its theoretical properties are known, it is fast and easy to program, it has a long period, and, most important, it produces numbers with good statistical properties.[1,2] Unfortunately, published results of the application of the multiplicative method so far have been for computers with word lengths of 36 bits or larger. There is some speculation whether the multiplicative method will be an adequate method of random number generation for the IBM 360. A problem inherent within the method concerns the selection of a multiplier which will produce numbers with good statistical properties. From a theoretical standpoint it is known that the multiplier

1) should be of the form $8k\pm3$ for k a non-negative integer in order to ensure the maximum length of period
2) should not be close to a simple rational multiple of the modulus or serial correlation will result
3) should not be close to a simple rational multiple of the square root of the modulus or correlation of lag 2 will result

but, aside from these rules, little is known about why some multipliers produce numbers with better statistical properties than others. This problem is magnified by the 360's small word length of 32 bits and it is believed that random number generation on the 360 by the multiplicative method will be extremely sensitive to choice of a multiplier.

In an attempt to learn more about random number generation on the IBM 360 the multiplicative method was programmed for the 360. A total

of 1,500 different multipliers was generated randomly and used to generate sequences of pseudorandom numbers. Each sequence was statistically tested and the test results were compared. Integer arithmetic was used to generate fixed point numbers and floating point numbers on the unit interval were obtained by setting to zero the first eight bits of each integer number and inserting the appropriate exponent. In order that the bit patterns for the integer random numbers and floating point random numbers be identical for the statistical tests the integers were stored as twenty four bit results with the first eight bits of each word set to zero. The modulus of the method thus was reduced to 2^{24} and since the starting value used was odd and all multipliers were of the form $8k\pm3$ for k a non-negative integer the period of all the generators tested was the maximum possible of 2^{22}. To test the 1,500 random multipliers the decimal number 2173 was arbitrarily selected as the starting value and for each multiplier a sequence of 5,000 or 10,000 numbers was generated. The frequency test and serial correlation test were performed on each sequence of generated numbers and for both tests a Chi-Square statistic and the associated Chi-Square probability (i.e., the probability that if the numbers are random a Chi-Square variable will exceed this test statistic) were calculated.

Of the 1,500 multipliers tested the ten which had the largest Chi-Square probabilities were selected for further testing. These ten multipliers were compared in three different ways with the nine multipliers which gave the lowest Chi-Square probabilities and for which the hypothesis of uniform randomness was rejected. First, since a multiplier could be as large as $2^{24}-1$ it could have as many as eight decimal digits and it was thought that larger decimal numbers were probably better

multipliers than smaller ones. However, some of the five-digit numbers were better multipliers than the eight-digit numbers so this idea had to be rejected. Coveyou[3] stated that the multiplier should not have a small number of 1's and it was conjectured by this author that the best multipliers probably had an equal number of 0's and 1's in the binary representation. After tallying 0's and 1's it appeared that this idea also had to be rejected. It was then suggested that by looking at the octal digits of the multipliers it might be observed that certain octal digits were present or were lacking in particular digit positions of the ten acceptable multipliers. This hypothesis again proved false and so the search for finding rules to predict when a multiplier would be a good one was concluded.

Next it was desired to learn more about the kinds of generators that were produced by using the ten selected multipliers as defined in the previous paragraph. Passing tests like the frequency and serial correlation tests is necessary but certainly not sufficient for the acceptance of a generator so each multiplier was subjected to an extensive set of tests. In all, sixteen tests were performed on the first 10,000 numbers generated with starting value 2173. The tests are described as follows:

TEST 1 - Frequency test on the generated numbers with 100 divisions of the unit interval.

TEST 2 - Serial correlation test. Successive 2-tuples of numbers were taken as points in the unit square. The unit square was divided into 100 equal cells and the frequency of 9,999 pairs was tallied.

TEST 3 - Successive 3-tuples of numbers were taken as points in the unit

cube. The unit cube was divided into 1,000 equal cells and the frequency of 9,998 triples was tallied.

TEST 4 - Consecutive, non-overlapping, 2-tuples of numbers were taken as points in the unit square. The unit square was divided into 100 equal cells and the frequency of 5,000 pairs was tallied.

TEST 5 - Consecutive, non-overlapping, 3-tuples of numbers were taken as points in the unit cube. The unit cube was divided into 1,000 equal cells and the frequency of 3,333 triples was tallied.

TEST 6 - Consecutive, non-overlapping, 4-tuples of numbers were taken as points in the unit cube in four space. The unit cube in four space was divided into 10,000 equal cells and the frequency of 2,500 4-tuples was tallied.

TEST 7 - Frequency test on the eight octal digit positions of each generated integer number.

TEST 8 - Poker test.

TEST 9 - Test of runs up and down.

TEST 10 - Test of runs above and below the mean.

TEST 11 - The maximum number in consecutive, non-overlapping, 2-tuples of numbers was calculated. The unit interval was divided into 100 equal cells and the frequency of the 5,000 maximums was tallied.

TEST 12 - The maximum number in consecutive, non-overlapping, 3-tuples of numbers was calculated. The unit interval was divided into 100 equal cells and the frequency of the 3,333 maximums was tallied.

TEST 13 - The maximum number in consecutive, non-overlapping, 4-tuples of numbers was calculated. The unit interval was divided into 100 equal cells; frequency of the 2,500 minimums was tallied.

For each test the calculated results for the generated sequence of numbers were compared with results that would have been expected from a truly random sequence of numbers by a Chi-Square test. The results of the Chi-Square test are given in TABLE 1. Each column of the table represents a particular multiplier, shown in decimal at the head of the column, and each row represents a particular test. The numbers listed represent the Chi-Square probability or percentage probability of exceeding the calculated Chi-Square statistic. The appearance of ϵ in the table means that the Chi-Square probability was less than .05 and that the hypothesis of uniform randomness was rejected. The eight probabilities listed for TEST 7 are the results of the frequency test performed separately on each of the eight octal digit positions of the 24 bit random integers. The first number is for bits 22-24 (counting beginning from the right end of the word), the second number bits 19-21, and the eighth number bits 1-3. It is obvious that the Chi-Square probability for the eighth digit position, bits 1-3, will be extremely low since the least significant bit must always be a 1 because the generated numbers are odd. The two tests on the digits, tests 7 and 8, are not as important as the other tests because most applications require numbers rather than digits. In this regard, the manner in which TABLE 1 is viewed depends largely on the particular application for which a generator is needed. If an application requires randomness of consecutive 2-tuples then the results of tests 4, 11, and 14 indicate that multiplier 9 should not be chosen. If, instead, good randomness of consecutive 3-tuples is needed then tests 5, 12, and 15 indicate that any of the multipliers is adequate. Tests 6, 13, and 16 show that for randomness of consecutive 4-tuples multiplier 10

TABLE 1

	-1- 13651723	-2- 12316123	-3- 50515	-4- 29355	-5- 54427
TEST 1	.94	.94	.98	.97	.99
TEST 2	.90	.92	.98	.97	.89
TEST 3	.93	.57	.98	.41	.76
TEST 4	.99	.68	.87	.99	.57
TEST 5	.79	.55	.48	.53	.42
TEST 6	.80	.27	.48	.33	.83
TEST 7	.95 .42 .51 .99 .99 .99 .99 ϵ	.85 .76 .16 .97 .99 .99 .99 ϵ	.76 .55 .42 .99 .99 .99 .99 ϵ	.81 .77 .98 .99 .99 .99 .99 ϵ	.69 .50 .58 .99 .99 .99 .99 ϵ
TEST 8	.08	.68	.07	.79	.27
TEST 9	ϵ	.07	.35	.56	.11
TEST 10	.26	.40	.93	.76	.90
TEST 11	.97	.96	.97	.83	.40
TEST 12	.94	.57	.62	.75	.95
TEST 13	.65	.86	.96	.29	.18
TEST 14	.53	.05	.98	.37	.92
TEST 15	.27	.30	.96	.76	.99
TEST 16	.90	.13	.88	.72	.74

TABLE 1 (Continued)

	-6- 223227	-7- 220915	-8- 220615	-9- 3739387	-10- 1459251
TEST 1	.95	.98	.94	.96	.96
TEST 2	.93	.89	.93	.88	.98
TEST 3	.52	.39	.88	.81	.60
TEST 4	.44	.89	.51	.72	.91
TEST 5	.48	.66	.93	.92	.34
TEST 6	.76	.90	.97	.62	.95
TEST 7	.59	.83	.72	.84	.54
	.55	.55	.34	.63	.93
	.65	.89	.66	.09	.38
	.99	.93	.99	.99	.99
	.99	.99	.99	.99	.99
	.99	.99	.99	.99	.99
	.99	.99	.99	.99	.99
	ϵ	ϵ	ϵ	ϵ	ϵ
TEST 8	.15	.44	.41	ϵ	.92
TEST 9	.85	.32	.84	ϵ	.20
TEST 10	.68	.14	.27	.28	.89
TEST 11	.52	.71	.70	ϵ	.40
TEST 12	.60	.42	.96	.22	.57
TEST 13	.54	.46	.66	.06	.41
TEST 14	.19	.98	.06	.85	.15
TEST 15	.70	.84	.73	.48	.21
TEST 16	.48	.70	.35	.17	ϵ

should not be chosen. Some applications require uniform numbers that oscillate randomly among themselves. This property of the generated numbers is tested for by tests 9 and 10 and the results of these tests show that multipliers 1 and 9 should not be used for this application.

Although the ten carefully chosen multipliers of TABLE 1 gave good results it must be remembered that the 10,000 numbers tested in each case comprise a very small sample of the 4,194,304 possible numbers in the period of the generator. Consequently, further tests were performed using different sample sizes and different starting values. TABLE 2 shows the effect of a change in starting value. Each row represents a particular decimal starting value shown at the left of each row and each column represents a particular decimal multiplier shown at the head of the column. The numbers listed in the table are the Chi-Square probabilities for the frequency test performed on a sample of 10,000 numbers generated with each particular starting value. TABLE 3 shows the effect of a change in sample size. Each row represents a particular size sample shown at the left of the row and each column represents a particular multiplier shown in decimal at the head of the column. The numbers listed in the table are the Chi-Square probabilities for the frequency test performed on particular size samples generated with a starting value of 2173. As the statistics in TABLES 2 and 3 show, when one repeats the same test on different samples one should get a range of Chi-Square values with the corresponding Chi-Square probabilities distributed uniformly. Since the interpretation of the statistics does not vary drastically with a change in sample size or starting value there is no reason to believe that the original tests on samples of 10,000 numbers with starting value 2173 were biased.

TABLE 2

	-1- 13651723	-2- 12316123	-3- 50515	-4- 29355	-5- 54427	-6- 223227	-7- 220915	-8- 220651	-9- 3739387	-10- 1459251
1	.76	.42	.69	.46	.95	.98	.31	.30	.88	.28
21	.39	.60	.38	.52	.52	.21	.61	ϵ	.39	ϵ
321	ϵ	.63	.12	.64	.82	.56	.64	ϵ	.97	.54
4321	.57	.92	.74	.39	.48	.29	.87	.51	.59	.42
54321	.78	ϵ	.66	.23	.62	.49	.55	.92	.59	.41
654321	.92	.59	.45	.48	.15	.36	.61	.54	ϵ	.68
7654321	.39	.33	ϵ	.93	ϵ	.73	.61	.13	.32	.94
87654321	.34	.06	.75	.79	.19	.84	.78	.61	.84	.19

TABLE 3

	-1- 13651723	-2- 12316123	-3- 50515	-4- 29355	-5- 54427	-6- 223227	-7- 220915	-8- 220651	-9- 3737387	-10- 1459251
5000	.91	.49	.99	.62	.90	.44	.88	.97	.96	.97
10000	.94	.94	.98	.97	.99	.95	.98	.94	.96	.96
15000	.93	.79	.67	.79	.76	.85	.95	.85	.83	.87
20000	.84	.52	.42	.14	.63	.53	.87	.75	.98	.78
25000	.72	.77	.29	.13	.37	.54	.96	.67	.95	.10
30000	.63	.57	.64	.20	.26	.40	.91	.66	.99	.25
35000	.75	.83	.73	.55	.17	.30	.95	.47	.99	.15
40000	.48	.84	.86	.24	.11	.37	.80	.94	.88	.07

This battery of tests was next used to learn more about the IBM random number generator for the 360, RANDU. There had been some hesitation to use RANDU because its statistical properties were not known. The same sixteen tests were applied to the first 10,000 numbers generated by RANDU with starting value 2173 and the Chi-Square probabilities are listed in TABLE 4 under multiplier 11. The results were theoretically predictable. The generator RANDU uses the multiplicative method with the modulus 2^{31} and the multiplier 65539. Note, however, that 65539 is $2^{16} + 3$. IBM made a mistake in choosing the multiplier so close to the square root of the modulus and this accounts for the extremely high correlation among 3-tuples as shown by the results of TEST 3.

One of the most recent developments in the random number generation field is a paper by Marsaglia[4] in which he reports that all multiplicative generators have a defect in that when successive n-tuples of numbers $(x_1,\ldots,x_n)$, $(x_2,\ldots,x_{n+1}),\ldots$ are plotted in the unit n-cube all the points will fall into a small number of parallel hyperplanes. This defect is not too serious since most users will not select random numbers in this manner anyway. If an application does require randomness of successive n-tuples, however, some improved methods of generation have been suggested. One of these methods, suggested earlier by MacLaren and Marsaglia,[5] involves the mixing of two separate multiplicative generators. This method was programmed for the 360 and produced random numbers $x_1,\ldots,x_n$ in the following manner. A list of 128 numbers $u_1,\ldots,u_{128}$ was produced by the relation $u_{i+1} \equiv a_1 u_i \pmod{2^{24}}$. Another relation $v_{i+1} \equiv a_2 v_i \pmod{2^{24}}$ was used to generate a second set of numbers $v_1,\ldots,v_n$. Bits 18-24 of v_k were used to select the k'th random number,

TABLE 4

	RANDU -11- 65539	MIXING -12- 220651 13651723
TEST 1	.67	.94
TEST 2	.38	.58
TEST 3	ε >.85	.86
TEST 4	.85	.90
TEST 5	.31	.46
TEST 6	.07	.91
TEST 7	.32 .93 .20 .99 .99 .99 .99	.96 .61 .51 .99 .99 .99 .99
TEST 8	.74	.09
TEST 9	.21	.20
TEST 10	.26	.50
TEST 11	.64	.70
TEST 12	.71	.97
TEST 13	.81	.21
TEST 14	.97	.34
TEST 15	.71	.64
TEST 16	.37	.68

x_k, from the list of the u's and this position was then refilled with v_{k+1}. The two multipliers selected for a_1 and a_2 were multipliers 8 and 1 respectively. The sixteen tests were performed on the first 10,000 numbers generated with starting value $u_0 = 2173$ and $v_0 = 2173$ and the Chi-Square probabilities are listed in TABLE 4 under the heading multiplier 12. Though this method eliminates the inherent non-randomness of successive n-tuples in the multiplicative method it requires 128 storage locations and is slower than the multiplicative method. The time for generation of one random number for multipliers 1-10 and RANDU was .0022 hundredths of a second, but the generation time for the Marsaglia mixing process was .0031 hundredths of a second. With the speed and storage capacities of modern-day computers, however, these two problems should not be serious for most users.

In conclusion, a battery of programs was written to provide stringent statistical tests for a random number generator. A group of random number generators was devised, each one meeting the needs of a different type of application. It is hoped that the user will consult TABLES 1-4 and then select the generator which best satisfies his requirements. In the likely event of a further breakthrough in the field of random number generation the battery of statistical programs will undoubtedly be of assistance in testing future random number generators.

FOOTNOTES

1. Coveyou and MacPherson, J. Assoc. Comp. Mach. 14. serial prop

2. Hull and Dobell, J. Assoc. Comp. Mach. 11.

3. Coveyou and MacPherson, 14.

4. Marsaglia, Proc. N. A. S. 61. correl

5. MacLaren and Marsaglia, J. Assoc. Comp. Mach. 12. maxima

BIBLIOGRAPHY

1. Coveyou, R. R. and MacPherson, R. D., "Fourier Analysis of Uniform Random Number Generators," J. Assoc. Comp. Mach. 14 (1967), pp. 100-119.

2. Good, I. J., "The Serial Test for Sampling Numbers and Other Tests for Randomness," Proc. Camb. Phil. Soc. 49 (1953), pp. 276-284.

3. Green, Jr., B. F., Smith, J. E., Klem, L., "Empirical Tests of an Additive Random Number Generator," J. Assoc. Comp. Mach. 6 (1959), pp. 527-536.

4. Greenberger, M. "Notes on a New Pseudo-Random Number Generator," J. Assoc. Comp. Mach. 8 (1961), pp. 163-166.

5. Hull, T. E. and Dobell, A. R., "Random Number Generators," SIAM Review 4 (1962), pp. 230-254.

6. Hull, T. E. and Dobell, A. R., "Mixed Congruential Random Number Generators for Binary Machines," J. Assoc. Comp. Mach. 11 (1964) pp. 31-40.

7. International Business Machines Corporation, Random Number Generation and Testing, Reference manual C20-8011 (New York, 1959).

8. Jannson, B., Random Number Generators, (Victor Pettersons Bokindustri Aktiebolag, Stockholm, 1966).

9. Kendall, M. G. and Babington-Smith, B., "Randomness and Random Sampling Numbers," J. Roy. Stat. Soc. 101 (1938), pp. 147-166.

10. Kendall, M. G. and Babington-Smith, B., Tables of Random Sampling Numbers, Tracts for Computers, no. 24 (Cambridge, 1939).

11. Kendall, M. G., and Stuart, A., The Advanced Theory of Statistics, (Griffin, London, 1958), I.

12. Kermack, W. O. and McKendrick, A. G., "Some Distributions Associated with a Randomly Arranged Set of Numbers," Proc. Roy. Soc. Edinburgh 57 (1937), pp. 332-376.

13. Lehmer, D. H., "Mathematical Models in Large-Scale Computing Units," Annals Comp. Laboratory, Harvard Univ. 26 (1951), pp. 141-146.

14. Levene H. and Wolfowitz, J., "The Covariance Matrix of Runs Up and Down," Annals Math. Stat. 15 (1944), pp. 58-69.

15. MacLaren, M. D. and Marsaglia, G., "Uniform Random Number Generators," J. Assoc. Comp. Mach. 12 (1965), pp. 83-88.

16. Marsaglia, G., "Random Numbers Fall Mainly in the Planes," Proc. N. A. S. 61 (1968), pp. 25-28.

17. Ore, O., Number Theory and Its History, (McGraw-Hill, New York, 1948).

18. Rand Corporation, A Million Random Digits with 100,000 Normal Deviates, (Free Press, Glencoe, 1955).

19. Richtmyer, R. D., "Monte Carlo Methods," Nuclear Reactor Theory, Am. Math. Soc., Proc. Symposium Appl. Math. 11 (1961), pp. 190-205.

20. Rotenberg, A., "A New Pseudo-Random Number Generator," J. Assoc. Comp. Mach. 7 (1960), pp. 75-77.

21. Student, "The Probable Error of a Mean," Biometrika 6 (1908), pp. 1-25.

22. Tippett, L. H. C., Random Sampling Numbers, Tracts for Computers, no. 15 (Cambridge, 1927).

APPENDIX

SUBROUTINE NAME *** RANB

IDENTIFICATION *** Random number generator

PURPOSE *** Generates a sequence of uniformly distributed random floating point numbers on the unit interval (0,1) and uniformly distributed random integers on the interval $(0,2^{22})$. The period of the generated numbers is 2^{22}.

OTHER SUBROUTINES USED *** None

USAGE ***
```
REAL*4 RN(NX)
INTEGER *4 IRN(NX)
.
.
.
CALL RANB (NSTART,N,RN,IRN)
```

Where the parameters in the calling sequence are:

NSTART - The starting value. It should be an odd positive integer.

N - The number of random numbers to be generated, NX $\geqslant$ N.

RN - The one dimensional array in which the N floating point random numbers are stored by RANB. Its contents may be arbitrary upon entry.

IRN - The one dimensional array in which the N integer random numbers are stored by RANB. Its contents may be arbitrary upon entry.

Note: When a particular multiplier is desired, the card labelled MULT in the program should be changed to the correct multiplier.

```
RANB     BEGIN
         L       3,4(1)          GETS ADDRESS OF N
         L       3,0(3)          PUTS N IN REG. 3
         L       6,8(1)          GETS ADDRESS OF RN(1) IN REG. 6
         L       8,12(1)         GETS ADDRESS OF IRN(1) IN REG. 8
         L       2,0(1)          GETS ADDRESS OF STARTING VALUE
         L       5,0(2)          PUTS STARTING VALUE IN REG. 5
         LR      10,5            SAVES THE STARTING VALUE
         LR      0,3
         L       11,MULT
         SR      7,7
LOOP     MR      4,11
         N       5,ZOOT          ZEROS OUT THE FIRST 8 BITS OF THE INTEGER
         ST      5,0(8,7)        STORES THE INTEGER IN THE ARRAY IRN
         O       5,CHAR          CONVERT TO FLOATING POINT
         ST      5,TEMP
         LE      0,TEMP          GETS THE RANDOM NUMBER IN A FLTING PT REG.
         AE      0,ZERO          NORMALIZES THE RANDOM NUMBER
         STE     0,TEMP
         L       5,TEMP          GETS THE NUMBER BACK IN GENL. REG. 5
         ST      5,0(6,7)        PUTS RANDOM NUMBER IN THE ARRAY RN
         L       5,0(8,7)        PUTS THE INTEGER IN REG. 5
         LA      7,4(7)
         BCT     0,LOOP
         ST      10,0(2)         RESTORES THE STARTING VALUE
         LEAVE
ZERO     DC      E'0.0'
CHAR     DC      X'40000000'     CHARACTERISTIC
ZOOT     DC      X'00FFFFFF'
MULT     DC      X'00035FF3'
TEMP     DS      1F
         END
```

SUBROUTINE *** UNINUM

IDENTIFICATION *** Frequency test for random numbers

PURPOSE *** Determines how close a set of random numbers are to being uniformly distributed by dividing the unit interval into 100 subintervals, counting the frequency of random numbers in each subinterval and comparing with expected results by a Chi-Square test.

OTHER SUBROUTINES USED *** PROLIM

USAGE *** REAL*4 RN (NX)

```
      .
      .
      .
      CALL UNINUM (N,RN,PROB)
```

Where the parameters in the calling sequence are:

N - The number of random numbers to be tested, NX $\geq$ N.

RN - The one dimensional array containing the N random numbers to be tested.

PROB - The outputed value of the Chi-Square probability. It is a number between 0 and 1 which indicates the probability that a Chi-Square variable will exceed the calculated test statistic.

```
      SUBROUTINE UNINUM(N,RN,PROB)
C     TEST OF UNIFORMITY OF NUMBERS WITH 100 DIVISIONS OF THE UNIT INTERVAL.
      REAL*4 RN(N),DIV(101)
      INTEGER*4 CELL(100)
      DIV(1)=0.0
      DO 5 I=2,101
5     DIV(I)=DIV(I-1)+.01
      DO 6 I=1,100
6     CELL(I)=0
      DO 8 I=1,N
      DO 7 J=1,100
      IF(RN(I).GE.DIV(J).AND.RN(I).LT.DIV(J+1))GO TO 8
7     CONTINUE
8     CELL(J)=CELL(J)+1
      EXPDIS=N/100.
      XNUM=0.0
      DO 9 I=1,100
9     XNUM=XNUM+(CELL(I)-EXPDIS)**2
      UNCSQ=XNUM/EXPDIS
      CALL PROLIM(99,UNCSQ,PROB)
      RETURN
      END
```

SUBROUTINE NAME *** FRESOC

IDENTIFICATION *** Frequency test on successive or consecutive n-tuples of random numbers.

PURPOSE *** Determines how close a set of n-tuples of random numbers are to being uniformly distributed. The n-tuples can be taken either successively or consecutively and n must be 1, 2, 3, or 4. The unit (interval, square, cube or cube in 4-space) is divided into (10, 10^2, 10^3, or 10^4) subintervals respectively depending on the value of n. The frequency of random numbers in each subinterval is tallied and compared with expected results by a Chi-Square test.

USAGE *** REAL*4 RN(NX)

```
      .
      .
      .
      CALL FRESOC (N,RN,NTUPLE,ISOC,CHISQ,PROB)
```

Where the parameters in the calling sequence are:

N - The number of random numbers to be used in the test, NX $\geq$ N.

RN - The one dimensional array containing the N random numbers to be tested.

NTUPLE - The size of the tuple, NTUPLE = 1,2,3,4.

ISOC - If ISOC = 1 then the N random numbers are to be divided into
N-NTUPLE+1 SUCCESSIVE TUPLES
~~N/NTUPLE consecutive, non-overlapping~~, tuples of numbers.
ISOC = 2 N/NTUPLE CONSEC. NON-OVERLAPPING

CHISQ - On exit from FRESOC CHISQ will contain the calculated Chi-Square statistic for the test. If ISOC = 1 the input value for CHISQ must be the CHISQ value obtained from a previous call to FRESOC with the NTUPLE value one less than the NTUPLE value for this call. If the NTUPLE value for this call is 1 then the input value for CHISQ is 0. If ISOC = 2 the input value for CHISQ may be arbitrary.

PROB - The outputed value of the Chi Square probability. It is a number between 0 and 1 which indicates the probability that a Chi-Square variable will exceed the calculated test statistic CHISQ.

```
      SUBROUTINE FRESUC(N,RN,NTUPLE,ISOC,UNCSQ,PROB)
C     FREQUENCY TEST ON SUCCESSIVE OR CONSECUTIVE N TUPLES
      REAL*4 RN(N),DIV(11)
      INTEGER*4 CELL(10,10,10,10),LPM(4)
      DIV(1)=0.0
      DO 5 I=2,11
5     DIV(I)=DIV(I-1)+0.1
      J=1
      K=1
      L=1
      DO 9 I=1,10
      DO 8 J=1,10
      DO 7 K=1,10
      DO 6 L=1,10
      CELL(I,J,K,L)=0
      IF(NTUPLE.EQ.1)GO TO 9
      IF(NTUPLE.EQ.2)GO TO 8
      IF(NTUPLE.EQ.3)GO TO 7
6     CONTINUE
7     CONTINUE
8     CONTINUE
9     CONTINUE
      DO 10 I=1,4
10    LPM(I)=1
      IF(ISOC.EQ.2)GO TO 20
      INCR=1
      IRMAX=N-NTUPLE+1
      GO TO 25
20    NUMTUP=N/NTUPLE
      IRMAX=NUMTUP*NTUPLE-NTUPLE+1
      INCR=NTUPLE
25    DO 50 IR=1,IRMAX,INCR
      DO 40 NT=1,NTUPLE
      IRP=IR+NT-1
      DO 30 J=1,10
      IF(RN(IRP).GE.DIV(J).AND.RN(IRP).LT.DIV(J+1))GO TO 40
30    CONTINUE
40    LPM(NT)=J
50    CELL(LPM(1),LPM(2),LPM(3),LPM(4))=CELL(LPM(1),LPM(2),LPM(3),LPM(4)
     1)+1
      IF(ISOC.EQ.2)GO TO 53
      EXPDIS=IRMAX/(10.**NTUPLE)
      GO TO 54
53    EXPDIS=NUMTUP/(10.**NTUPLE)
```

```
54      XNUM=0.0
        J=1
        K=1
        L=1
        DO 60 I=1,10
        DO 59 J=1,10
        DO 58 K=1,10
        DO 57 L=1,10
        XNUM=XNUM+(CELL(I,J,K,L)-EXPDIS)**2
        IF(NTUPLE.EQ.1)GO TO 60
        IF(NTUPLE.EQ.2)GO TO 59
        IF(NTUPLE.EQ.3)GO TO 58
57      CONTINUE
58      CONTINUE
59      CONTINUE
60      CONTINUE
        IF(ISOC.EQ.2)GO TO 70
        TEMCSQ=XNUM/EXPDIS
        UNCSQ=TEMCSQ-UNCSQ
        NDF=10**NTUPLE-10**(NTUPLE-1)
        GO TO 80
70      UNCSQ=XNUM/EXPDIS
        NDF=10**NTUPLE
80      CALL PROLIM(NDF,UNCSQ,PROB)
        RETURN   WRITE   /   /
        END
```

corrcl

SUBROUTINE NAME *** SERPLT

IDENTIFICATION *** Plotter routine for random number generator

PURPOSE *** Successive 2-tuples of random numbers are plotted as points in the unit square. Any preponderance of points in an area of the plot indicates serial correlation. Note that the numbers printed along each axis should be divided by 10 to obtain their actual values.

OTHER SUBROUTINES USED *** CCP1PL,CCP5AX

USAGE *** REAL*4 RN(NX)

```
      REAL*4 RN(NX)
        .
        .
        .
      CALL SERPLT(N,RN)
```

Where the parameters in the calling sequence are:

N - The number of random numbers to be used in the test, NX $\geqslant$ N.

RN - The one dimensional array containing the N random numbers to be tested.

```
      SUBROUTINE SERPLT(N,RN)
      REAL*4 RN(N),T(2),EDGE
      INTEGER*4 UP,DOWN,LAST(16,16)
      LGRID=16
C     THINK BASE LGRID
C     LGRID MUST BE POWER OF TWO
      DO 10 I=1,LGRID
      DO 10 J=1,LGRID
10    LAST(I,J)=0
      NM1=N-1
C     PUT POINTS IN BOXES
      DO 50 I=1,NM1
      LAB=LGRID*RN(I)+1
      LORD=LGRID*RN(I+1)+1
      LPREV=LAST(LAB,LORD)
      IF(LPREV.EQ.0)GO TO 40
      LDIFF=I-LPREV
      RN(I)=RN(I)+LDIFF
C     STORED BACK POINTER IN RN(I)
40    LAST(LAB,LORD)=I
50    CONTINUE
C     HAVE POINTS IN BOXES, NOW DECODE AND PLOT.
      T(1)=0.0
      T(2)=1.25
      CALL CCP1PL(0.5,0.5,-3)
      CALL CCP5AX(0.0,0.0,21HSECOND MEMBER OF PAIR,21,8.0,90.0,T)
      CALL CCP5AX(0.0,0.0,20HFIRST MEMBER OF PAIR,-20,8.0,0.0,T)
      UP=3
      DOWN=2
      EDGE=0.01
      DO 100 LAB=1,LGRID
      DO 99 LORD=1,LGRID
      I=LAST(LAB,LORD)
60    IF(I.EQ.0)GO TO 99
      JX=RN(I)
      JY=RN(I+1)
      RN(I)=RN(I)-JX
      FY=RN(I+1)-JY
      XC=8.0*RN(I)
      YC=8.0*FY
      XL=YC-EDGE
      XM=XC+EDGE
      YL=YC-EDGE
      YM=YC+EDGE
      CALL CCP1PL(XL,YL,UP)
      CALL CCP1PL(XM,YM,DOWN)
      CALL CCP1PL(XM,YL,UP)
      CALL CCP1PL(XL,YM,DOWN)
      LDIFF=JX
      IF(LDIFF.EQ.0)GO TO 99
      I=I-LDIFF
      GO TO 60
99    CONTINUE
100   CONTINUE
      RETURN
      END
```

SUBROUTINE NAME *** POKDIG

IDENTIFICATION *** Poker test and frequency test on the octal digits of a set of random integers.

PURPOSE *** The 24 bit random integers are divided into 8 octal digits each. In the poker test the first 5 octal digits of each integer are treated as a poker hand without regard to suit. The frequencies of certain poker hands are tallied and compared with expected results by a Chi-Square test. In the frequency test the frequency of each octal digit in each of the 8 digit positions is tallied and compared with expected results by a Chi-Square test.

OTHER SUBROUTINES USED *** FREPOK,PROLIM

USAGE ***
```
REAL*4 PROB1(8)
INTEGER *4 IRN (NX)
.
.
.
CALL POKDIG(N,IRN,IFOP,PROB1,PROB2)
```

Where the parameters in the calling sequence are:

N - The number of random integers to be used in the test, NX > N.

IRN - The one dimensional array containing the N random integers to be tested.

IFOP - If IFOP=1 then the frequency test is to be performed. If IFOP = 2 then the poker test is to be performed.

PROB1 - If IFOP = 1 then PROB1(I), I = 1,...,8 contains the Chi-Square probability of the frequency test for the i'th digit position where the 1st digit position is bits 22-24 (counting from the right end of the word), the 2nd digit position is bits 19-21,..., the 8'th digit position is bits 1-3. The Chi-Square probability is a number between 0 and 1 which indicates the

probability that a Chi-Square variable will exceed the calculated test statistic. If IFOP = 2, then PROB1(I), I = 1,...,8 will be arbitrary upon exit from POKDIG.

PROB2 - If IFOP = 1 then PROB2 will be arbitrary upon exit from POKDIG. If IFOP = 2 then PROB2 contains the Chi-Square probability for the poker test. It is a number between 0 and 1 which indicates the probability that a Chi-Square variable will exceed the calculated test statistic.

```
      SUBROUTINE POKDIG(N,IRN,IFOP,PROB1,PROB2)
C     FREQUENCY TEST AND POKER TEST ON THE DIGITS
      REAL*4 EXPPOK(7),PROB1(8)
      INTEGER*4 OCT(8,8),POK(7),IRN(N)
      DO 3 I=1,8
      DO 3 J=1,8
3     OCT(I,J)=0
      DO 4 I=1,7
4     POK(I)=0
      CALL FREPOK(N,IRN,OCT,POK,IFOP)
      IF(IFOP.EQ.2)GO TO 20
      EXPFRE=N/8.
      DO 15 J=1,8
      XNUM=0.0
      DO 14 I=1,8
14    XNUM=XNUM+(OCT(I,J)-EXPFRE)**2
      CHIFRE=XNUM/EXPFRE
15    CALL PROLIM(7,CHIFRE,PROB1(J))
      GO TO 1000
20    EXPPOK(1)=0.205078*N
      EXPPOK(2)=0.512695*N
      EXPPOK(3)=0.153809*N
      EXPPOK(4)=0.102539*N
      EXPPOK(5)=0.017090*N
      EXPPOK(6)=0.008545*N
      EXPPOK(7)=0.000244*N
      CHIPOK=0.0
      DO 26 I=1,7
26    CHIPOK=CHIPOK+(POK(I)-EXPPOK(I))**2/EXPPOK(I)
      CALL PROLIM(6,CHIPOK,PROB2)
1000  RETURN
      END
```

SUBROUTINE NAME *** FREPOK

IDENTIFICATION *** A subroutine called by POKDIG

PURPOSE *** The 24 bit random integers are divided into 8 octal digits each. In the poker test the first 5 octal digits of each integer are treated as a poker hand without regard to suit. The frequencies of certain poker hands are tallied by FREPOK. In the frequency test the frequency of each octal digit in each of the 8 digit positions is tallied by FREPOK.

OTHER SUBROUTINES USED *** None

USAGE *** INTEGER*4 IRN(NX),OCT(8,8),POK(7)

```
      INTEGER*4 IRN(NX),OCT(8,8),POK(7)
          .
          .
          .
          .
      CALL FREPOK(N,IRN,OCT,POK,IFOP)
```

Where the parameters in the calling sequence are:

N - The number of random integers to be used in the test, NX $\geqslant$ N.

IRN - The one dimensional array containing the N random integers to be tested.

OCT - If IFOP = 1 this two dimensional array is set to zero on entry to FREPOK. On exit from FREPOK OCT(I,J),I,J = 1,...,8 contains the frequency of the octal digit I-1 in the J'th digit position where bits 22-24 are the 1st digit position,..., bits 1-3 are the 8th digit position. If IFOP = 2 the contents of this array are arbitrary on entry and exit from FREPOK.

POK - If IFOP = 2 then POK(I), I = 1,...,7 is set to zero on entry to FREPOK. On exit from FREPOK POK contains the frequencies of 7 poker hands where:

POK(1) = frequency of bust

POK(2) = frequency of one pair

POK(3) = frequency of two pair

POK(4) = frequency of 3 of a kind

POK(5) = frequency of a full house

POK(6) = frequency of 4 of a kind

POK(7) = frequency of 5 of a kind

If IFOP = 1 then the contents of this array are arbitrary on entry and exit from FREPOK.

IFOP - If IFOP = 1 then the frequency of digits is to be tallied. If IFOP = 2 then the frequency of poker hands is to be tallied.

```
FREPOK   BEGIN
         L      5,0(1)
         L      5,0(5)       PUTS N IN REG. 5
         L      10,8(1)      PUTS ADDRESS OF OCT(1,1) IN REG. 10
         L      11,4(1)      PUTS ADDRESS OF IRN(1) IN REG. 11
         L      4,16(1)
         L      4,0(4)       PUTS IFOP IN REG. 4
         L      14,ONE
         LA     0,31(14)
NEWRN    CR     4,14
         BC     8,FRESRT
         SR     2,2
         SR     3,3          THIS INITIALIZES THE DIST ARRAY TO ZERO
INIT     ST     3,DIST(2)    THE DIST ARRAY STORES THE NUMBER
         LA     2,4(2)       OF TIMES THAT EACH OCTAL DIGIT OCCURRED
         CR     2,0          IN THE FIRST FIVE OCTAL DIGITS OF
         BC     7,INIT       A RANDOM NUMBER.
FRESRT   L      3,0(11)      PUTS IRN(I) IN REG. 3
         SLDL   2,8          GETS RID OF INITIAL ZEROS OF IRN(I)
         SR     9,9          REG. 9 TELLS WHICH OCTAL POSITION
NEWOCT   SR     7,7          REG. 7 TELLS WHICH OCTAL DIGIT
         SR     2,2          CLEARS REG. 2
         SLDL   2,3          MOVES AN OCTAL DIGIT INTO REG. 2
LOOP     SR     2,14
         BC     4,HERE       FOUND WHICH OCTAL DIGIT IT IS
         LA     7,1(7)
         BC     15,LOOP
HERE     M      6,FOUR
         CR     4,14
         BC     8,FRETST     FREQUENCY TEST ONLY
         L      6,DIST(7)
         LA     6,1(6)
         ST     6,DIST(7)
         LA     9,1(9)
         C      9,FIVE
         BC     8,POKTST
         BC     15,NEWOCT
FRETST   MR     8,0
         AR     7,9          ADDRESS MODIFICATION OF OCT(1,1)
         L      2,0(10,7)    PUTS OCT(I,J) IN REG. 2
         LA     2,1(2)
         ST     2,0(10,7)
         DR     8,0          RESTORES REG. 9
         LA     9,1(9)
         C      9,EIGHT
         BC     8,GETRN
         BC     15,NEWOCT    MORE OCTAL DIGITS IN IRN(I)
POKTST   L      2,12(1)      PUTS ADDRESS OF POK(1) IN REG. 2
         SR     6,6          REG. 6 TELLS WHETHER 3 OF A KIND
         SR     7,7          REG. 7 TELLS HOW MANY PAIRS
         SR     8,8          REG. 8 USED TO INCREMENT THE DIST ARRAY
```

```
MORE     L      9,DIST(8)
         C      9,FIVE           TEST FOR 5 OF A KIND
         BC     7,B1
         L      3,24(2)
         LA     3,1(3)           INCREMENT 5 OF A KIND,POK(7)
         ST     3,24(2)
         BC     15,GETRN
B1       C      9,FOUR           TEST FOR 4 OF A KIND
         BC     7,B2
         L      3,20(2)
         LA     3,1(3)           INCREMENT 4 OF A KIND,POK(6)
         ST     3,20(2)
         BC     15,GETRN
B2       C      9,THREE          TEST FOR 3 OF A KIND
         BC     7,B3
         LA     6,1(6)
B3       C      9,TWO            TEST FOR A PAIR
         BC     7,B4
         LA     7,1(7)
B4       LA     8,4(8)
         CR     8,0
         BC     8,REST           NO MORE DIST ELEMENTS TO TEST
         BC     15,MORE
REST     CR     6,14             IS THERE 3 OF A KIND?
         BC     7,B6
         CR     7,14             IS THERE A PAIR?
         BC     7,B5
         L      3,16(2)
         LA     3,1(3)           INCREMENT FULL HOUSE,POK(5)
         ST     3,16(2)
         BC     15,GETRN
B5       L      3,12(2)
         LA     3,1(3)           INCREMENT 3 OF A KIND,POK(4)
         ST     3,12(2)
         BC     15,GETRN
B6       C      7,TWO
         BC     7,B7
         L      3,8(2)
         LA     3,1(3)           INCREMENT 2 PAIR,POK(3)
         ST     3,8(2)
         BC     15,GETRN
B7       CR     7,14
```

```
         BC      7,B8
         L       3,4(2)
         LA      3,1(3)          INCREMENT 1 PAIR,POK(2)
         ST      3,4(2)
         BC      15,GETRN
B8       L       3,0(2)
         LA      3,1(3)          INCREMENT A BUST,POK(1)
         ST      3,0(2)
GETRN    LA      11,4(11)
         BCT     5,NEWRN         MORE RANDOM NUMBERS?
         LEAVE
ONE      DC      F'1'
TWO      DC      F'2'
THREE    DC      F'3'
FOUR     DC      F'4'
FIVE     DC      F'5'
EIGHT    DC      F'8'
DIST     DS      8F
         END
```

SUBROUTINE NAME *** RNSUAD

IDENTIFICATION *** Test of runs up and down on a set of random numbers.

PURPOSE *** If $x_1, \ldots x_N$ are the set of random numbers then

$$x_{n-1} \leqslant x_n > x_{n+1} > \cdots > x_{n+k} \leqslant x_{n+k+1} \quad \text{is a run down of length } k,$$

$$\text{and } x_{n-1} > x_n \leqslant x_{n+1} \leqslant \cdots \leqslant x_{n+k} > x_{n+k+1} \quad \text{is a run up of length } k.$$

This test involves tallying runs up and down of different lengths and comparing with expected results by a Chi-Square test.

OTHER SUBROUTINES USED *** PROLIM

USAGE ***
```
REAL*4 RN(NX),EXPRUN(100)
INTEGER*4 RUNLTH(100)
.
.
.
CALL RNSUAD(N,RN,EXPRUN,RUNLTH,PROB)
```

Where the parameters in the calling sequence are:

N - The number of random numbers to be tested, $NX \geqslant N$.

RN - The one dimensional array containing the N random numbers to be tested.

EXPRUN - Internal array used by RNSUAD, its contents are arbitrary on entry to RNSUAD. On exit from RNSUAD it contains the expected frequencies of runs of lengths 1,...m where m is the maximum run length found in the N random numbers.

RUNLTH - Internal array used by RNSUAD, its contents are arbitrary on entry to RNSUAD. On exit from RNSUAD it contains the actual frequencies of runs of lengths 1,...m where m is the maximum run length found in the N random numbers.

PROB - The outputed value of the Chi-Square probability. It is a number between 0 and 1 which indicates the probability that a Chi-Square variable will exceed the calculated test statistic.

```
      SUBROUTINE RNSUAD(N,RN,EXPRUN,RUNLTH,PROB)
C     TEST OF RUNS UP AND DOWN
      REAL*4 RN(N),EXPRUN(1)
      INTEGER*4 RUNLTH(1)
      MAXLTH=0
      DO 2 I=1,100
2     RUNLTH(I)=0
      LASTZ=0
      LAST1=0
      I=1
      IF(RN(I).GT.RN(I+1))GO TO 10
5     I=I+1
      IF(I.EQ.N)GO TO 13
      IF(RN(I).LE.RN(I+1))GO TO 5
      K=I-LAST1-1
      LASTZ=I-1
      NSWICH=1
      GO TO 15
10    I=I+1
      IF(I.EQ.N)GO TO 14
      IF(RN(I).GT.RN(I+1))GO TO 10
      K=I-LASTZ-1
      LAST1=I-1
      NSWICH=0
      GO TO 15
13    K=N-LAST1-1
      GO TO 15
14    K=N-LASTZ-1
15    RUNLTH(K)=RUNLTH(K)+1
      IF(K.GT.MAXLTH)MAXLTH=K
      IF(I.EQ.N)GO TO 23
      IF(NSWICH.EQ.0)GO TO 5
      GO TO 10
23    DO 25 I=1,MAXLTH
      IP3=I+3
      FACT=1.0
      DO 24 J=1,IP3
24    FACT=FACT*J
      ISQ=I**2
25    EXPRUN(I)=2.*(N*(ISQ+3.*I+1.)-(ISQ*I+3.*ISQ-I-4.))/FACT
      IMAX=MAXLTH
      DO 50 I=1,IMAX
      II=IMAX-I+1
      IF(RUNLTH(II).GE.5)GO TO 51
      RUNLTH(II-1)=RUNLTH(II-1)+RUNLTH(II)
      EXPRUN(II-1)=EXPRUN(II-1)+EXPRUN(II)
50    MAXLTH=MAXLTH-1
51    CHISQ=0.0
      DO 60 I=1,MAXLTH
60    CHISQ=CHISQ+(RUNLTH(I)-EXPRUN(I))**2/EXPRUN(I)
      NDF=MAXLTH-1
      CALL PROLIM(NDF,CHISQ,PROB)
      RETURN
      END
```

SUBROUTINE NAME *** RNSABM

IDENTIFICATION *** Test of runs above and below the mean on a set of random numbers.

PURPOSE *** If $x_1, \ldots x_N$ are the set of random numbers then

$x_{n+1} \leq 1/2$, $x_{n+2} \leq 1/2, \ldots, x_{n+k} \leq 1/2$ is a run below the mean of length k and $x_{n+1} > 1/2$, $x_{n+2} > 1/2, \ldots, x_{n+k} > 1/2$ is a run above the mean of length k. This test involves tallying runs above and below the mean of different lengths and comparing with expected results by a Chi-Square test.

OTHER SUBROUTINES USED *** PROLIM

USAGE ***
```
REAL*4 RN(N), EXPRUN(100)
INTEGER *4 RUNLTH(100)
.
.
.
CALL RNSABM(N,RN,EXPRUN,RUNLTH,PROB)
```

Where the parameters in the calling sequence are:

N - The number of random numbers to be tested, NX $\geq$ N.

RN - The one dimensional array containing the N random numbers to be tested.

EXPRUN - Internal array used by RNSABM, its contents are arbitrary on entry to RNSABM. On exit from RNSABM it contains the expected frequencies of runs of lengths 1,...m where m is the maximum run length found in the N random numbers.

RUNLTH - Internal array used by RNSABM, its contents may be arbitrary on entry to RNSABM. On exit from RNSABM it contains the actual frequencies of runs of lengths 1,...,m where m is the maximum run length found in the N random numbers.

PROB - The outputed value of the Chi-Square probability. It is a number between 0 and 1 which indicates the probability that a Chi-Square variable will exceed the calculated test statistic.

```
      SUBROUTINE RNSABM(N,RN,EXPRUN,RUNLTH,PROB)
C     TEST OF RUNS ABOVE AND BELOW THE MEAN
      REAL*4 RN(N),EXPRUN(1)
      INTEGER*4 RUNLTH(1)
      MAXLTH=0
      DO 2 I=1,100
2     RUNLTH(I)=0
      LASTZ=0
      LAST1=0
      I=1
      NP1=N+1
      IF(RN(I).GT.0.5)GO TO 10
5     I=I+1
      IF(I.EQ.NP1)GO TO 13
      IF(RN(I).LE.0.5)GO TO 5
      K=I-LAST1-1
      LASTZ=I-1
      NSWICH=1
      GO TO 15
10    I=I+1
      IF(I.EQ.NP1)GO TO 14
      IF(RN(I).GT.0.5)GO TO 10
      K=I-LASTZ-1
      LAST1=I-1
      NSWICH=0
      GO TO 15
13    K=N-LAST1
      GO TO 15
14    K=N-LASTZ
15    RUNLTH(K)=RUNLTH(K)+1
      IF(K.GT.MAXLTH)MAXLTH=K
      IF(I.EQ.NP1)GO TO 33
      IF(NSWICH.EQ.0)GO TO 5
      GO TO 10
33    DO 35 I=1,MAXLTH
35    EXPRUN(I)=(N-I+3.)*2.**(-I-1)
      IMAX=MAXLTH
      DO 50 I=1,IMAX
      II=IMAX-I+1
      IF(RUNLTH(II).GE.5)GO TO 51
      RUNLTH(II-1)=RUNLTH(II-1)+RUNLTH(II)
      EXPRUN(II-1)=EXPRUN(II-1)+EXPRUN(II)
50    MAXLTH=MAXLTH-1
51    CHISQ=0.0
      DO 60 I=1,MAXLTH
60    CHISQ=CHISQ+(RUNLTH(I)-EXPRUN(I))**2/EXPRUN(I)
      NDF=MAXLTH-1
      CALL PROLIM(NDF,CHISQ,PROB)
      RETURN
      END
```

GAP

SUBROUTINE NAME *** MAXMIN

IDENTIFICATION *** Frequency test on maximums or minimums of n-tuples of a set of random numbers. If W is the maximum of the n numbers in an n-tuple then $X = W^n$ is uniformly distributed; if W is the minimum of the n numbers in an n-tuple then $Y = 1-(1-W)^n$ is uniformly distributed. In this test the unit interval is divided into 100 subintervals and the frequency of X's and Y's is tallied and compared against expected results by a Chi-Square test.

OTHER SUBROUTINES USED *** UNINUM

USAGE *** REAL*4 RN(NX),MXOMN(MX)

```
        .
        .
        .
        CALL MAXMIN(N,RN,NTUPLE,MXOMN,IXON,PROB)
```

Where the parameters in the calling sequence are:

N - The number of random numbers to be tested, NX $\geq$ N.

RN - The one dimensional array containing the N random numbers to be tested.

NTUPLE - The number of random numbers in a tuple. Note that the tuples are taken consecutively from the N random numbers.

MXOMN - An internal array used by MAXMIN, MX $\geq$ N/NTUPLE. Its contents are arbitrary on entry to MAXMIN; on exit from MAXMIN it contains the N/NTUPLE X's or N/NTUPLE Y's.

IXON - If IXON = 1 then the maximum of n-tuples is to be tested. If IXON = 2 then the minimum of n-tuples is to be tested.

PROB - The outputed value of the Chi-Square probability. A number between 0 and 1 which indicates the probability that a Chi-Square variable will exceed the calculated test statistic.

```
      SUBROUTINE MAXMIN(N,RN,NTUPLE,MXOMN,IXON,PROB)
      REAL*4 RN(N),MXOMN(1)
      IN=N/NTUPLE
      IRMAX=IN*NTUPLE-NTUPLE+1
      NTM1=NTUPLE-1
      I=1
      DO 40 J=1,IRMAX,NTUPLE
      MXOMN(I)=RN(J)
      IF(IXON.EQ.2)GO TO 20
      DO 10 K=1,NTM1
10    IF(RN(J+K).GT.MXOMN(I))MXOMN(I)=RN(J+K)
      MXOMN(I)=MXOMN(I)**NTUPLE
      GO TO 40
20    DO 30 K=1,NTM1
30    IF(RN(J+K).LT.MXOMN(I))MXOMN(I)=RN(J+K)
       MXOMN(I)=1.-(1.-MXOMN(I))**NTUPLE
40    I=I+1
      CALL UNINUM(IN,MXOMN,PROB)
      RETURN
      END
```

SUBROUTINE NAME *** PROLIM

IDENTIFICATION *** Chi-Square probability

PURPOSE *** Given a Chi-Square statistic X^2 and the number of degrees of freedom n, this subroutine calculates the probability that a Chi-Square variable will exceed X^2. For $n \leq 30$ this is done by evaluating

$$\int_{X^2}^{\infty} f(x)dx \text{ where } f(x) = \frac{x^{n/2-1}e^{-x/2}}{2^{n/2}\Gamma(n/2)}$$ is the Chi-Square probability

distribution. For $n > 30$ the quantity $\sqrt{2X^2} - \sqrt{2n-1}$ is normally distributed

so $$\int_{X^2}^{\infty} f(x)dx \text{ where } f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

is evaluated.

OTHER SUBROUTINES USED *** None

USAGE *** CALL PROLIM(NDF CHISQ,PROB)

Where the parameters in the calling sequence are:

NDF - The number of degrees of freedom, an input parameter.

CHISQ - The Chi-Square statistic X^2, an input parameter.

PROB - The Chi-Square probability, an output parameter. It is a number between 0 and 1 which indicates the probability that a Chi-Square variable will exceed the statistic CHISQ.

```
SUBROUTINE PROLIM(NDF,UNCSQ,PROB)
REAL*4 X(96),W(96)
FUNC1(T)=(T**((NDF-2)/2.))*EXP(-T/2.)
FUNC2(T)=EXP((-T**2)/2.)
X(01) = 00.01627674
X(02) = -0.01627674
X(03) = 00.04881298
X(04) = -0.04881298
X(05) = 00.08129749
X(06) = -0.08129749
X(07) = 00.11369585
X(08) = -0.11369585
X(09) = 00.14597371
X(10) = -0.14597371
X(11) = 00.17809688
X(12) = -0.17809688
X(13) = 00.21003131
X(14) = -0.21003131
X(15) = 00.24174315
X(16) = -0.24174315
X(17) = 00.27319881
X(18) = -0.27319881
X(19) = 00.30436494
X(20) = -0.30436494
X(21) = 00.33520852
X(22) = -0.33520852
X(23) = 00.36569686
X(24) = -0.36569686
X(25) = 00.39579764
X(26) = -0.39579764
X(27) = 00.42547898
X(28) = -0.42547898
X(29) = 00.45470942
X(30) = -0.45470942
X(31) = 00.48345797
X(32) = -0.48345797
X(33) = 00.51169417
X(34) = -0.51169417
X(35) = 00.53938810
X(36) = -0.53938810
X(37) = 00.56651041
X(38) = -0.56651041
X(39) = 00.59303236
X(40) = -0.59303236
```

X(41) = 00.61892584
X(42) = -0.61892584
X(43) = 00.64416340
X(44) = -0.64416340
X(45) = 00.66871831
X(46) = -0.66871831
X(47) = 00.69256453
X(48) = -0.69256453
X(49) = 00.71567681
X(50) = -0.71567681
X(51) = 00.73803064
X(52) = -0.73803064
X(53) = 00.75960234

X(54) = -0.75960234
X(55) = 00.78036904
X(56) = -0.78036904
X(57) = 00.80030874
X(58) = -0.80030874
X(59) = 00.81940031
X(60) = -0.81940031
X(61) = 00.83762351
X(62) = -0.83762351
X(63) = 00.85495903
X(64) = -0.85495903
X(65) = 00.87138850
X(66) = -0.87138850
X(67) = 00.88689451
X(68) = -0.88689451
X(69) = 00.90146063
X(70) = -0.90146063
X(71) = 00.91507142
X(72) = -0.91507142
X(73) = 00.92771245
X(74) = -0.92771245
X(75) = 00.93937033
X(76) = -0.93937033
X(77) = 00.95003271
X(78) = -0.95003271
X(79) = 00.95968829
X(80) = -0.95968829
X(81) = 00.96832682
X(82) = -0.96832682
X(83) = 00.97593917
X(84) = -0.97593917
X(85) = 00.98251726
X(86) = -0.98251726
X(87) = 00.98805412
X(88) = -0.98805412
X(89) = 00.99254390
X(90) = -0.99254390
X(91) = 00.99598184

```
X(92) = -0.99598184
X(93) = 00.99836437
X(94) = -0.99836437
X(95) = 00.99968950
X(96) = -0.99968950
W(01) = 0.03255061
W(02) = 0.03255061
W(03) = 0.03251611
W(04) = 0.03251611
W(05) = 0.03244716
W(06) = 0.03244716
W(07) = 0.03234382
W(08) = 0.03234382
W(09) = 0.03220620
W(10) = 0.03220620
W(11) = 0.03203445
W(12) = 0.03203445
W(13) = 0.03182875
W(14) = 0.03182875

W(15) = 0.03158933
W(16) = 0.03158933
W(17) = 0.03131642
W(18) = 0.03131642
W(19) = 0.03101033
W(20) = 0.03101033
W(21) = 0.03067137
W(22) = 0.03067137
W(23) = 0.03029991
W(24) = 0.03029991
W(25) = 0.02989634
W(26) = 0.02989634
W(27) = 0.02946108
W(28) = 0.02946108
W(29) = 0.02899461
W(30) = 0.02899461
W(31) = 0.02849741
W(32) = 0.02849741
W(33) = 0.02797000
W(34) = 0.02797000
W(35) = 0.02741296
W(36) = 0.02741296
W(37) = 0.02682686
W(38) = 0.02682686
W(39) = 0.02621234
W(40) = 0.02621234
W(41) = 0.02557003
W(42) = 0.02557003
W(43) = 0.02490063
W(44) = 0.02490063
W(45) = 0.02420484
W(46) = 0.02420484
```

```
W(47) = 0.02348339
W(48) = 0.02348339
W(49) = 0.02273706
W(50) = 0.02273706
W(51) = 0.02196664
W(52) = 0.02196664
W(53) = 0.02117293
W(54) = 0.02117293
W(55) = 0.02035679
W(56) = 0.02035679
W(57) = 0.01951908
W(58) = 0.01951908
W(59) = 0.01866067
W(60) = 0.01866067
W(61) = 0.01778250
W(62) = 0.01778250
W(63) = 0.01688547
W(64) = 0.01688547
W(65) = 0.01597056
W(66) = 0.01597056
W(67) = 0.01503872
W(68) = 0.01503872
W(69) = 0.01409094
W(70) = 0.01409094
W(71) = 0.01312822

W(72) = 0.01312822
W(73) = 0.01215160
W(74) = 0.01215160
W(75) = 0.01116210
W(76) = 0.01116210
W(77) = 0.01016077
W(78) = 0.01016077
W(79) = 0.00914867
W(80) = 0.00914867
W(81) = 0.00812687
W(82) = 0.00812687
W(83) = 0.00709647
W(84) = 0.00709647
W(85) = 0.00605854
W(86) = 0.00605854
W(87) = 0.00501420
W(88) = 0.0050142
W(89) = 0.00396455
W(90) = 0.00396455
W(91) = 0.00291073
W(92) = 0.00291073
W(93) = 0.00185396
W(94) = 0.00185396
W(95) = 0.00079679
W(96) = 0.00079679
PROB=0.0
```

```
      IF(NDF.GT.30)GO TO 20
      B=UNCSQ/2.
      S=0.0
      DO 10 J=1,96
      Y=B*X(J)+B
10    S=S+FUNC1(Y)*W(J)
      PROB=B*S+PROB
      DEN=(2.**(NDF/2.))*GAMMA(NDF/2.)
      PROB=1.-PROB/DEN
      RETURN
20    MOP=0
      B=SQRT(2.*UNCSQ)-SQRT(2.*NDF-1.)
      IF(B.GE.0.0)GO TO 25
      B=-B
      MOP=1
25    B=B/2.
      S=0.0
      DO 30 J=1,96
      Y=B*X(J)+B
30    S=S+FUNC2(Y)*W(J)
      PROB=B*S+PROB
      PROB=.3989422*PROB
      IF(MOP.EQ.1)GO TO 40
      PROB=.5-PROB
      RETURN
40    PROB=.5+PROB
      RETURN
      END
```